DIE HEXEN KÜCHE

von Peter Finzel



Tips, Kniffe und Programme für

ATARI® 400 600 800 XL

I M P R E S S U M

P. FINZEL: DIE HEXENKÜCHE, TIPS, INFOS UND PROGRAMME FÜR ATARI 400/600/800/XL

Peter Finzel
Bremer Str. 19
8510 Fürth/Bay.

Für eine Mitteilung eventueller Fehler sowie Anregungen ist der Autor jederzeit dankbar (Anschrift siehe oben).

1. Auflage 19S4

Anmerkung zu dieser elektronischen Ausgabe:

Herr Peter Finzel hat mir vor Jahren das Recht eingeräumt sein Buch "Die Hexenküche" auf meiner Webseite www.hintermueller.de in elektronischer Form zu veröffentlichen.

Die vorliegende elektronische Ausgabe unterscheidet sich etwas von der gedruckten Vorlage. Ich habe mir erlaubt das damals wohl auf einem Nadeldrucker im Blocksatz und mit fester Schriftweite gedruckte Buch in eine etwas angenehmer lesbare Form zu bringen, Überschriften nicht nur durch Unterstreichungen zu kennzeichnen, etc. Außerdem wurde die Schreibweise einigermaßen an die heute gültige Rechtschreibung angeglichen. Man hat sich einfach daran gewöhnt und das Lesen ist angenehmer und flüssiger, wenn man nicht ständig über „irgendetwas“ stolpert.

Dadurch sind an der ein oder anderen Stelle auch die Seitenzahlen etwas verrutscht. Inhaltlich entspricht diese elektronische Ausgabe jedoch vollständig dem Original. Einzig bei den abgedruckten Assemblerlistings habe ich die Listings der zum Buch gehörenden Diskette verwendet - soweit traue ich der OCR-Texterkennung dann doch nicht ☺

Daher fehlen im Gegensatz zur Originalausgabe die Angaben der Speicherbereiche auf der linken Seite des Listings. Der Funktion tut das aber keinen Abbruch.

INHALTSVERZEICHNIS

Vorwort

Es ist sicher keine Neuigkeit für Sie, dass Ihr Atari ein extrem leistungsfähiger und flexibler Computer ist. Neu ist aber, dass Sie mit der HEXENKÜCHE noch mehr aus Ihrem Computer herausholen können. Voraussetzung dazu sind ein paar BASIC-Kenntnisse und etwas guter Wille sich auch mit Maschinenprogrammen zu befassen. Bei der HEXENKÜCHE handelt es sich nicht etwa um ein neues Lehrbuch der 6502-Assemblersprāche, davon gibt es wahrlich schon genügend, sondern vielmehr um eine Sammlung von interessanten und nützlichen Tipps, Kniffen, BASIC- und Maschinenprogrammen. Da man komplizierte Sachverhalte bekanntlich durch Beispiele am besten verstehen kann, ist natürlich ein Lerneffekt nicht auszuschließen.

Alle Programme sind reichlich kommentiert und erklärt, so dass Sie die Programme schnell verstehen können. Für diejenigen unter Ihnen, die an der Programmierung von Spielprogrammen interessiert sind, enthält das Buch zwei Leckerbissen: Eine einbaufertige Musikroutine und ein komplettes Sound-Entwicklungssystem, das Sie direkt in Ihren eigenen Programmen einsetzen können.

Die zweite Hälfte des Buches widmet sich mehr den Systemprogrammen. Unter anderem erfahren Sie, wie Sie Diskettenoperationen in Maschinensprache programmieren und wie Sie effektiv mit dem leistungsstarken OS/A+ DOS (auch als CP/A+ bekannt) arbeiten können.

Die Hexenküche wäre aber nicht komplett, wenn nicht einige besondere Bonbons enthalten wären. Wussten Sie schon, dass Ihr Computer außer den bisherigen 17 Graphikmodi noch drei weitere beherrscht? Außerdem lesen Sie, wie Sie ein Textfenster in den Graphikstufen 9 bis 11 einrichten können, nur eines von vielen BASIC-Programmen, aus dem Sie sehen, wie BASIC mit Maschinenunterprogrammen unterstützt wird. BASIC—Programmierer kommen daher auch voll auf Ihre Kosten. Abgerundet wird die HEXENKÜCHE von einigen Abschnitten, die Ihnen zusätzliche Hintergrundinformationen zu den Programmen geben.

Noch ein paar Bemerkungen:

1.) An dieser Stelle vielen Dank an Rudolf Baser, Richard Stahl und nicht zuletzt an Petra Kolbe, die zum Gelingen dieses Buches freundschaftlich beigetragen haben.

2.) Obwohl die Programme auf einem 48K-Atari entwickelt und getestet wurden, laufen die meisten auch mit 16K Speicher. Allein das Musik— und das Soundprogramm wurden bewusst in höhere Speicherbereiche gelegt, damit dem Benutzer ein Maximum an BASIC Speicherplatz bleibt. Aus diesem Grund sind diese beiden Programme erst ab 48K lauffähig.

3.) NOCH EINE DRINGENDE BITTE:
Kopieren des Buches und der zugehörigen Diskette schadet nicht nur dem Autor, sondern auch Ihnen. Bedenken Sie, dass die Entstehung des vorliegenden Buches etwa ein halbes Jahr in Anspruch genommen hat, und nur unter der Voraussetzung durchführbar war, dass ein Markt dafür vorhanden ist. Raubkopierer zerstören sich und anderen die Chance, auch in Zukunft interessante Publikationen zu bekommen. In diesem Sinne
bitte ich Sie, weder die HEXENKÜCHE noch die Programmdiskette zu vervielfältigen. Vielen Dank!

Fürth/Bay., August 1984 Peter Finzel

Programmierung des VBI's

Wenn Sie Graphik- und Soundeffekte programmieren möchten, dann sollten Sie über eine äußerst nützliche und leistungsfähige Einrichtung Ihres Computers Bescheid wissen: den VBI (Vertical Blank Interrupt), zu Deutsch etwa 'Unterbrechung in der vertikalen Austastlücke'. Was kann man darunter verstehen? Der Bezeichnung können Sie entnehmen, dass es sich dabei um einen sogenannten 'Interrupt' handelt. Darunter versteht man eine von außen an den Prozessor heran getragene Aufforderung sein momentanes Programm (man spricht hier speziell von einem Vordergrundprogramm) zu unterbrechen, um eine 'Interruptroutine' aufzurufen. Speziell beim VBI wird 50-mal pro Sekunde ein Interrupt ausgelöst, der die VBI-Routine anstößt. Um erklären zu können, warum diese Einrichtung für graphische Anwendungen so interessant ist, muss man kurz betrachten, wie ein Monitor- bzw. Fernsehbild zustande kommt. Das Bild wird mit Hilfe eines Elektronenstrahles, beginnend in der linken oberen Ecke des Schirmes, Rasterzeile für Rasterzeile auf den Schirm geschrieben. Ist der Strahl schließlich in der rechten unteren Ecke des Schirmes angekommen, so muss er wieder in die linke obere Ecke zurückgelangen. In diesem Zeitraum ist der Elektronenstrahl natürlich nicht auf dem Schirm sichtbar, und just am Anfang dieser 'Vertikalen Austastung' wird der VBI angestoßen.

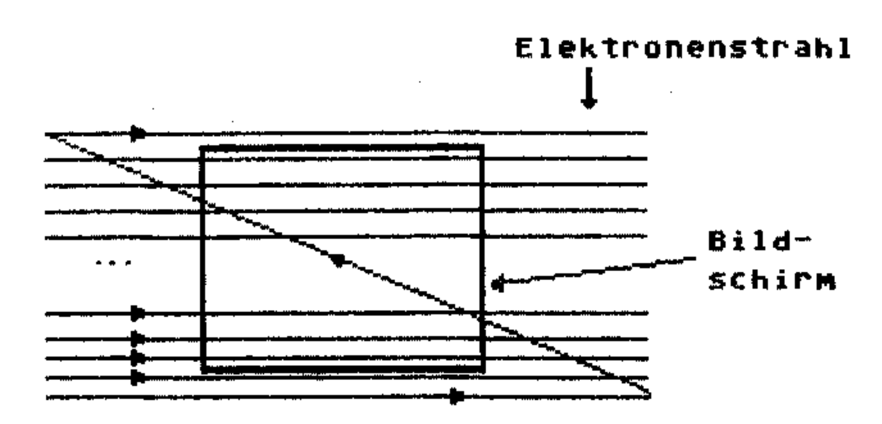


Bild 1: Elektronenstrahl im Fernsehgerät

Jetzt wird erkennbar, warum der VBI für graphische Anwendungen so wichtig ist: Veränderungen der Graphik im VBI können nicht am Bildschirm als Störungen sichtbar werden, da ja gerade dann kein Bild ausgegeben wird. Dadurch werden Playerbewegungen fließend, ruckfreies Scrolling überhaupt erst möglich und dergleichen mehr. Der VBI ist aber nicht nur graphischen Anwendungen vorbehalten, sondern kann selbstverständlich auch für alle Art von Aufgaben, die periodisch anfallen, eingesetzt werden, als da wären: Zeitmessungen, Abfrage der Joysticks oder auch zur Geräusch- und Musikerzeugung, mehr dazu lesen Sie in den nächsten Abschnitten.

Der VBI und das Betriebssystem

Wenn ein VBI ausgelöst wurde, so verzweigt *der* Programmablauf zuerst in das Betriebssystem. Hier werden zunächst die Registerinhalte auf den Stack gerettet, und anschließend wird indirekt durch den RAM-Vektor VVBLKI (bei $222/223 L/H) gesprungen. Dieser Vektor zeigt, jedenfalls solange er nicht verändert wurde, auf die VBI Routine des Betriebssystems, welche in zwei Stufen unterteilt ist:

Stufe 1: Hier wird die 'Uhr' in den Speicherzellen $12-14 weitergezählt, das Farbumschalten (Attractmodus) geprüft, und der System Timer 1 bearbeitet.

Stufe 2: Kopieren der Schattenregister in die zugehörigen Hardwareadressen, lesen der Joystickports usw.

Im Anschluss an Stufe 2 wird durch den RAM-Vektor VVBLKD (bei $224/225) gesprungen, der (wiederum im Normalfall) auf ein kleines Programm XITVBV zur Beendigung des Interrupts zeigt.

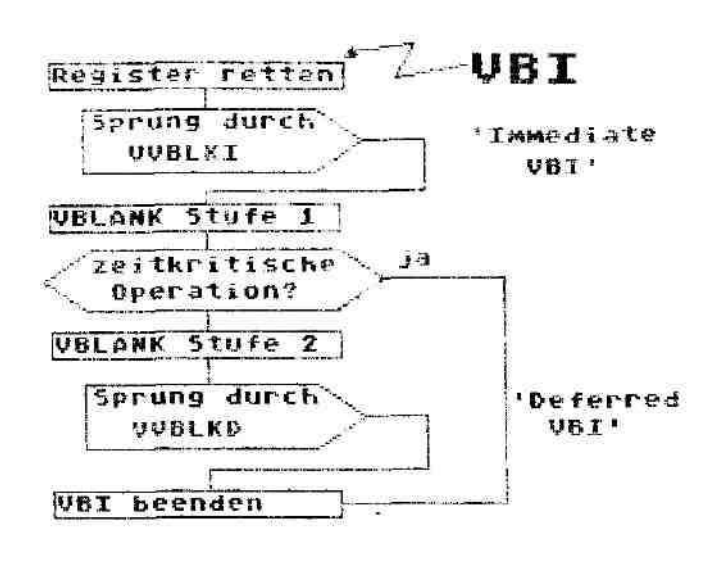


Bild 2: Ablauf des VBI

Die Stufe 2 und der Sprung durch VVBLKD werden aber nur ausgeführt, solange keine 'zeitkritische Bedingung' (z. B. Disk-Ein/Ausgabe, siehe unten) vorliegt.

Warum gleich zwei VBIs?

Ihre Chance liegt im Abändern der RAM-Vektoren, so dass sie auf eigene Programme zeigen. Wie Sie sehen, gibt es gleich zwei solcher Vektoren, mit denen Sie ein Programm in den VBI einfügen können: Den sog. 'Immediate VBI' und den 'Deferred VBI'. Wozu gleich zwei Möglichkeiten? Das ist schnell erklärt: Der VBI findet auch während I/O Operationen, z.B. während des Ladens eines Programmes von Diskette, statt. Da aber in einer solchen Situation nicht viel überflüssige Zeit vorhanden ist, wird der VBI abgekürzt, es wird nur der Sprung durch den 'Immediate VBI' und Stufe 1 der ROM-VBI-Routine ausgeführt. Danach verzweigt der Programmablauf sofort zu XITVBV, so dass Stufe 2, in der ja das doch etwas zeitaufwendigere Kopieren der Schattenregister stattfindet, und der Sprung durch VVBLKD entfällt.

Konkret bedeutet das:

1.) Wenn Ihr Programm auch während einer I/O—Operation aktiv bleiben soll, so müssen Sie es in den 'Immediate VBI' einfügen. Interessant ist dies z.B. für einen Programmvorspann, der während des Ladevorganges des eigentlichen Programmes den Programmtitel blinkend am Bildschirm anzeigt oder gar einfache Animation vornimmt.

2.) In allen anderen Fällen nimmt man den 'Deferred VBI', der abgeschaltet wird, wenn I/O läuft. Ist die I/O-Operation beendet, so schaltet er sich automatisch wieder ein.

Wie viel Rechenzeit haben Sie?

Im 'Immediate VBI' sollte Ihr VBI–Programm nicht länger als 400 Mikrosekunden sein, während des Vertical Blanks entspricht das ungefähr 650 Maschinenzyklen. Im 'Deferred VBI' können Sie längere VBI-Programme einfügen, die aber 20.000 Mikrosekunden keinesfalls überschreiten dürfen, denn nach dieser Zeit wird ja bereits der nächste VBI ausgelöst. Sie sollten außerdem zwei Gesichtspunkte im Auge behalten:

in diesen 20.000 Mikrosekunden ist auch die Zeit enthalten, in der schon wieder das nächste Bild ausgegeben wird. Die eigentliche 'Vertikale Austastlücke' ist schon nach *ca.* 4500 Mikrosekunden (bei europäischen PAL-ATARIs!) vorbei. In dieser Zeit können immerhin ca. 7400 Maschinenzyklen verarbeitet werden, damit kann man eine ganze Menge anfangen. Dabei ist allerdings zu bedenken, dass die ROM-VBI Routine davon ca. 900-1000 Zyklen verbraucht.

die Rechenzeit, die im VBI verbraucht wird, geht natürlich von der des Vordergrundprogrammes ab. Wenn Sie daher ein sehr zeitaufwendiges Programm im VBI haben und dabei noch BASIC laufen lassen, werden Sie feststellen, dass BASIC zu 'schleichen' beginnt.

Noch ein Wort zur Dauer des Vertical Blanks: Hier besteht ein wesentlicher Unterschied zwischen (amerikanischen) NTSC-ATARIs und (europäischen) PAL-ATARIs. Während die Bildwechselfrequenz bei der PAL-Version 50 Hz beträgt, arbeiten die amerikanischen ATARIs mit 60 Hz. Dadurch wird die dem VBI zur Verfügung stehende Zeit rigoros verkürzt, statt satter 4500 Mikrosekunden dauert der VBI nur noch 1400 Mikrosekunden (ca. 2300 Maschinenzyklen). Wenn Ihre Programme auch auf amerikanischen Systemen ohne Störungen laufen sollen, dann sollten Sie dieses Limit nicht überschreiten. Im Normalfall können Sie sich trotzdem etwas länger Zeit nehmen, da jede Displaylist mit drei mal acht Leerzeilen beginnen sollte, in denen ja ebenfalls keine Bildinformation ausgegeben wird. In dieser Zeit, nochmal ca. 1500 Mikrosekunden (2500 Zyklen), können Sie ebenfalls Graphikänderungen vornehmen, die nicht als Störungen sichtbar werden.

Fazit: Beim PAL-ATARI haben Sie 7400-1000+2500 = 8900 Zyklen
Beim NTSC-ATARIs 2300-1000+2500 = 3800 Zyklen
(VBI-Zyklen minus ROM-VBI plus leere D.-List)

Wie man die Zyklen eines Maschinenprogrammes auszählt, können Sie im Abschnitt 'Wie schnell ist Ihr Atari' nachlesen.

So kommen Sie heran:

Bleiben wir fürs erste beim 'Deferred VBI'. Wie Sie bereits wissen, gibt es im RAM zwei Speicherzellen bei

VVBLKD : $224,225 (L,H)
durch die nach der Bearbeitung des ROM-VBIs gesprungen wird. Dieser Vektor zeigt auf eine kleine Routine namens XITVBV ($E462), in welcher der VBI beendet wird. Wenn Sie diesen Vektor nun auf Ihr eigenes Programm zeigen lassen, und nach dessen Ablauf nach XITVBV springen, so wird Ihr Programm Bestandteil des VBIs.

Es ist nun nicht ratsam VVBLKD mit einigen STA-Befehlen (oder gar POKE in BASIC) zu ändern, denn stellen Sie sich folgendes vor:

Gerade nachdem Sie das niederwertige Byte bei $224 geändert haben, wird ein Interrupt ausgelöst. Da das Programm sofort unterbrochen wird, findet der VBI einen Vektor vor, der sich aus dem alten High-Byte und dem neuen Low-Byte zusammensetzt, und daher

mit größter Wahrscheinlichkeit zum Abstürzen des Computers führt.

Zum Glück bietet das Operating System des Computers eine einfache Lösung dieses Problems an: Das X-Register wird mit dem höherwertigen Byte der Anfangsadresse der VBI–Routine geladen, das Y-Register mit dem niederwertigen Byte. Um den Deferred VBI anzusprechen, ist im Akkumulator eine 7 abzulegen. Schließlich ist die ROM–Routine zum Eintragen des neuen Vektors aufzurufen: Unterprogrammsprung nach SETVBV ($E45C).

```
LDX #06    ;     High-Byte z.B. für Page 6
LDY #00    ;     Low-Byte
LDA #7     ;     für VBI
JSR SETVBV ;     neuen Vektor eintragen
```

Diese Routine würde ein Programm, welches ab $0600 (die berühmte PAGE 6...) im Speicher steht, in den VBI einbinden. Ihr VBI-Programm selbst muss die Beendigung des Interrupts beinhalten, der letzte ausgeführte Befehl muss

```
JMP XITVBV ; XITVBV := $E462
```

lauten. Damit wird das Unterbrechungsprogramm beendet, **und** das Hauptprogramm fortgesetzt.

Zusammenfassend müssen Sie also folgende Schritte unternehmen um ein Maschinenprogramm in den Deferred VBI einzufügen:

1.) Das Programm im Speicher ablegen, letzter ausgeführter Befehl muss JMP XITVBV ($E462) sein.

2.) High-Byte der Anfangsadresse ins X-Register. Low-Byte ins Y-Register

3.) Akku mit 7 laden (für Deferred VBI!)

4.) Unterprogrammsprung JSR SETVBV ($E45C)

Immediate VBI

Im Prinzip gilt das oben Gesagte auch für den 'Immediate VBI'. Unterschiedlich ist:

1.) der RAM-Vektor steht bei VVBLKI ($222,223 L,H)

2.) vor dem Aufruf von SETVBV muss der Akku mit '6' (anstatt der '7' beim Deferred VBI) geladen werden.

3.) der letzte Befehl Ihrer Routine muss mit JMP SYSVBV ($E45F) auf den Anfang der ROM-VBI Routine zeigen, vorausgesetzt, dass Sie diese Routine nicht überspringen wollen.

Ausschalten eines VBI–Programmes

funktioniert im Grunde genauso wie das Einschalten. Sie benutzen wieder die Routine SETVBV und laden damit den jeweiligen Vektor mit seinem ursprünglichen Wert.

```
Beim Deferred VBI:  LDX #XITVBV/256 ; XITVBV=$E462
                    LDY #XITVBV&255 ; LSB
                    LDA #7 ;    für Deferred VBI
                    JSR SETVBV

Beim Immediate VBI: LDX #SYSVBV/256 ; SYSVBV=$E45F
                    LDY #SYSVBV&255 ; LSB
                    LDA #6 ;    für Immediate VBI
                    JSR SETVBV
```

Übrigens: durch Drücken von SYSTEM RESET werden VBI–Routinen auch abgeschaltet.

ACHTUNG:

Um vor bösen Überraschungen sicher zu sein, sollten VBI-Routinen immer mit einem 'CLD'-Befehl beginnen, so dass der Prozessor in seine normale binäre Arbeitsweise geschaltet wird. Besonders wenn der VBI parallel zu BASIC läuft, befindet sich der 6502 zum Zeitpunkt des VBI-Aufrufes oft im BCD-Modus, und dieser Betriebszustand wird von der ROM-VBI-Routine nicht aufgehoben. Konsequenz: Additions- und Subtraktionsbefehle liefern andere Ergebnisse!

Mehrstimmige Musik – aber mit Hüllkurve

Falls Sie schon einmal versucht haben ein mehrstimmiges Musikprogramm in BASIC zu schreiben, dann wissen Sie sicher, dass es dabei einige Schwierigkeiten gibt: Das Timing ist schwierig, besonders bei kurzen Noten stimmt die Länge nicht, und die einzelnen Stimmen werden bei der Verwendung des SOUND-Befehles hörbar nacheinander angestoßen... Tatsächlich unmöglich wird es für BASIC, falls man versucht die Töne mit einer Hüllkurve zu versehen oder noch gleichzeitig graphische Abläufe auf dem Bildschirm darstellen will. Dass so etwas möglich ist, sehen Sie an einigen Spielprogrammen, wo mehrstimmige Musik während (!) des Programmablaufes erklingt. Das in diesem Abschnitt vorgestellte Maschinenprogramm zeigt Ihnen, wie's gemacht wird und auf eine recht elegante Art und Weise noch dazu...

Was kann das Musikprogramm?

Wollen wir zuerst einen Blick auf die Möglichkeiten werfen, die Ihnen mit einem solchen Programm offenstehen:

- mehrstimmige Musik - bis zu vier Stimmen gleichzeitig
- jede Stimme hat eine eigene programmierbare Hüllkurve
- Eingabe der Noten im Klartext (keine Zahlen !)
- bis zu 128 Noten pro Kanal spiel bar
- während die Musik erklingt, steht der Computer voll zu Ihrer Verfügung, d.h. BASIC oder natür1ich auch Maschinenprogramme können gleichzeitig laufen... Falls Sie besonders musikalisch sind können Sie Ihr Programm sogar zum Wohlklang der Musik editieren.

Wir machen Musik

Erreicht wird dies durch Benutzung des VBIs (s. Kapitel 'Der VBI'), wobei man gleich zwei Fliegen mit einer Klappe schlagen kann. Einerseits kann man durch die periodische Struktur des VBIs die Zeitdauer der Noten leicht programmieren, andererseits läuft das ganze Programm quasi 'nebenbei', so dass der Computer einstweilen andere Aufgaben erledigen kann.

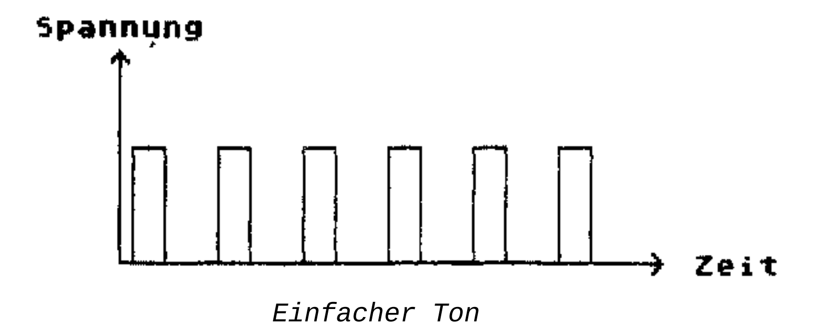
Hüllkurven

Im Vorausgehenden war jetzt schon von einer Hüllkurve die Rede - aber - was ist das eigentlich? Dazu einige Grundlagen: Ein einfacher Ton besteht aus einer Folge von Impulsen, die alle die gleiche Höhe (Amplitude) haben.

Ein Beispiel in BASIC für einen derartigen Ton ist

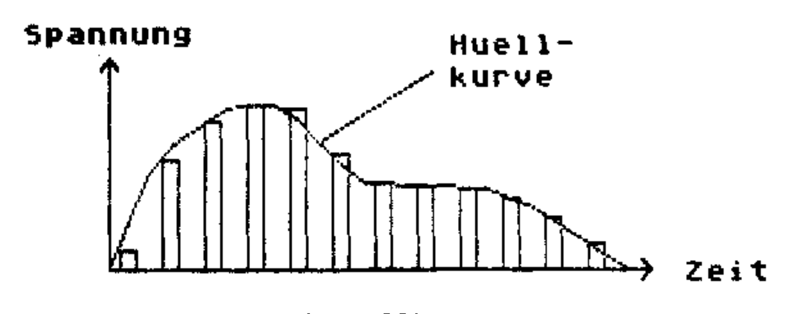
```
SOUND 0,50,10,8
```

wobei der letzte Parameter '8' des SOUND-Befehles ein Maß für die Lautstärke (und somit Amplitude) des Tones ist. Damit kann man durchaus Melodien spielen, die aber immer irgendwie nach einer Drehorgel klingen.



Einfacher Ton

Wesentlich interessanter klingt es, wenn man dem Ton eine Charakteristik, d.h. einen Anschlag und ein Abklingverhalten, in Form einer Hüllkurve gibt. Man versteht darunter ein schnelles Ändern der Lautstärke während der Ton erklingt.



Ton mit Hüllkurve

Sie sehen auch, woher der Name Hüllkurve kommt: Die Impulsspitzen werden sozusagen davon 'eingehüllt'.

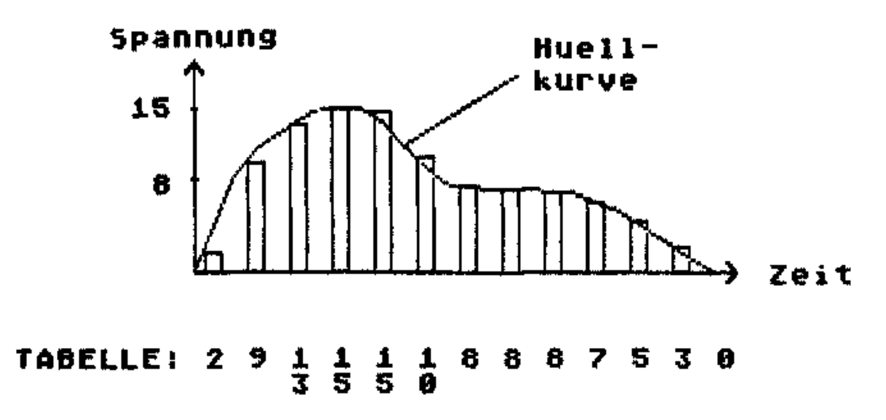
Ein Beispiel einer recht einfachen Hüllkurve in BASIC ist

```
FOR L=30 TO 0 STEP-1:SOUND 0,50,10,L/2:NEXT L
```

Haben Sie's bemerkt?

Es geht noch besser!

Wir wollen uns allerdings nicht auf so einfache Hüllkurven wie in dem kleinen BASIC-Beispiel beschränken. Zur Nachbildung von komplexen Hüllkurven werden die Lautstärkestützpunkte der Hüllkurve in Form einer Tabelle abgelegt.



Ablegen der Hüllkurve in einer Tabelle

Beim Anspielen eines Tones wird ein Zeiger auf den Anfang dieser Tabelle eingerichtet, der in jedem VBI weitergeschaltet wird. Der aktuelle Lautstärkewert wird aus der Tabelle entnommen und in die Volume–Bits 0–3 des AUDCx Registers eingetragen.

Damit ist man in der Lage Instrumente naturgetreuer nachbilden zu können. Der harte Anschlag eines Klaviers drückt sich in einem steileren Anstieg der Kurve aus, flötenartigen Klang erreicht man durch langsameres steigern der Lautstärke. An dieser Stelle ist Ihrer Kreativität nichts in den Weg gestellt. Experimentieren Sie ruhig mit den verschiedensten Hüllkurven. Es lohnt sich!

Eingabe der Noten:

Die Noteneingabe lässt sich in einem Assemblerprogramm recht elegant gestalten, indem die 'Intelligenz' des Assemblers gleich zur Übersetzung der Notenbezeichnungen in den jeweiligen Frequenzwert benutzt wird.

Alle Noten sind als so genannte 'EQUATES' vordefiniert, so nennt man Konstante, denen mit dem '='-Befehl ein Wert zugewiesen wurde. (Der Name kommt daher, dass in vielen anderen Assemblern 'EQU' anstatt '=' verwendet wird). Die Noten werden dann mit Ihrer Bezeichnung in eine Zeile eingetragen, die jeweils mit einem '.BYTE'–Befehl beginnt. Dabei gibt es folgendes zu beachten:

Die Noten sind, soweit es möglich war, in gängiger Schreibweise definiert, Etwas abweichend sind die Equates der Halbtöne:

Note:		einzutragen als:
C#	=	CIS
D#	=	DIS
F#	=	FIS
G#	=	GIS
A#	=	AIS
und PAUSE	=	P

Nach der Notenbezeichnung ist noch die Oktave zu vermerken, zulässig sind hier die Werte 3 und 4 (plus der 5 für das C), wobei 3 die tiefste Oktave ist. Die Frequenzwerte wurden übrigens der originalen ATARI Tabelle entnommen. Nach dem Notensymbol ist durch Komma getrennt die Notendauer einzutragen, für die ebenfalls schon Equates vordefiniert sind. Allerdings wurden hier englische Bezeichnungen gewählt, da sich mit den deutschen Schwierigkeiten ergaben: 'A' für Achtel ist in den meisten Assemblern ein reserviertes Wort für 'Akku'.

Symbol :	Notenlänge
H :	halbe Note (Half)
Q :	viertel Note (Ouarter)
E :	achtel Note (Eighth)
S :	sechzehntel Note (Sixteenth)

Punktierte Noten können einfach durch die Addition von zwei aufeinander folgenden Notenlängen erzielt werden:

H+Q : punktierte halbe Note

Die Eingabe einer kompletten Note (Beispiel: C#, 3. Oktave. Viertelnote) sieht damit so aus:

```
.BYTE CIS3,Q
```

Selbstverständlich können Sie auch mehrere Noten in eine Zeile eingeben. Der Übersicht wegen ist es zu empfehlen, für jeweils einen Takt auch eine Zeile zu verwenden, z.B.

```
.BYTE C3,H,E3,Q,P,Q (P steht für Pause)
```

bei 4/4-Takt. Die Zuordnung der Noten zu einer Stimme erfolgt dadurch, dass vor dem jeweils ersten .BYTE-Befehl einer Stimme der Label STIMM1 für Stimme 1, STIMM2 für Stimme 2 usw. vermerkt wird. Sie können sich dabei am Beispielprogramm orientieren. Beendet wird eine Notenfolge für eine Stimme durch die Eintragung des Wortes 'ENDE' anstelle eines Notenwertes. Schreiben Sie statt 'ENDE' den Befehl 'START', so wird die Stimme kontinuierlich von Anfang an wiederholt. Falls Sie weniger als 4 Stimmen verwenden wollen, dann schreiben Sie '.BYTE ENDE' gleich hinter dem jeweiligen Stimmlabel, der nicht verwendet werden soll. Zum Beispiel mit

```
STIMM4 .BYTE ENDE
```

würde Stimme 4 ruhig bleiben. Verwenden Sie in diesem Fall nie 'START', dadurch würde sich der Rechner 'aufhängen'!

Eingabe der Hüllkurve

Zu jeder Stimme muss eine eigene Hüllkurve eingegeben werden. Sie müssen dazu nur, wie eingangs beschrieben, die Zahlenwerte der aufeinander folgenden Lautstärkewerte der Reihe nach eintragen, wie dies im Bild 3 dargestellt ist. Die Werte werden wieder in .BYTE-Befehlen untergebracht und mit den Labels HUELL1 bis HUELL4 den einzelnen Stimmen zugeordnet (siehe Beispiel im Programm). Abgeschlossen wird jede Hüllkurve, genau wie bei den Notenzeilen, mit 'ENDE'.

Jetzt geht's los

Falls Sie glücklicher Besitzer der zum Buch erhältlichen Diskette sind, können Sie sich gleich ein Demo anhören. Sie brauchen nur (mit eingestecktem BASIC-Modul) das Programm

```
RUN "D:MUSIK.BAS"
```

aufrufen, und schon erklingt ein Musikstück als Demo. Diesen sogenannten 'BASIC-Loader' finden Sie im Anschluss an das Musik-Maschinenprogramm abgedruckt, in dieser Form können Sie das Musikprogramm auch in ein eigenes Programm übernehmen. Zwei wesentliche Nachteile dieser Methode sollen aber nicht verschwiegen werden: Zum einen benötigt der BASIC-Loader eine geraume Weile bis er das Maschinenprogramm samt Musik-Daten in den Speicher 'gepoked' hat, schlimmer ist aber, dass Sie im BASIC-Programm keine anderen Melodien eingeben können. Der BASIC-Loader ist daher nur geeignet, wenn Sie eine einbaufertige Musik-Routine brauchen oder nur eben mal hören wollen, was das Musikprogramm kann.

Falls Sie eine andere Melodie eingeben möchten, so sollten Sie sich mit dem Assemblerlisting befassen. Sie können es in (fast) jeden Assembler für den ATARI eintippen, jedenfalls solange er den Standard 6502-Assemblercode verwendet. Auf jeden Fall funktionieren: Editor/Assembler Cartridge, EASMD, MAC/65. Auf der oben genannten Diskette finden Sie das Quell-File unter dem Namen VBIMUSIK.SRC.

Das Musik-Maschinenprogramm

Der Aufruf des Musikprogrammes erfolgt über eine Sprungleiste am Anfang des Programmes (ab $9800, 38912 dez.), so dass der BASIC-Befehl zum Einschalten der Musik

```
X=USR(38912)
```

lauten muss. Abgeschaltet wird ebenfalls über diese Sprungleiste, der JMP-Befehl auf die passende Routine steht vier Bytes vom Programmanfang entfernt. Entsprechender BASIC-Aufruf:

```
X=USR(38916)
```

Natürlich können Sie das Musikprogramm auch in Maschinensprache aufrufen, dazu müssen Sie aber die PLA-Befehle in der Sprungleiste, die nur für den USR-Befehl in BASIC benötigt werden, übergehen. Ein entsprechendes Maschinenprogramm könnte so aussehen:

```
MEIN = $9801 Musik einschalten
MAUS = $9805 Musik ausschalten
...
...
JSR MEIN Musik erklingt ...
...
JSR MAUS jetzt Musik wieder aus!
...
```

Der Ablauf des Musik-Maschinenprogrammes

Der erste JMP-Befehl der Sprungleiste, der, wie Sie gerade gesehen haben, zum Einschalten des Musikprogrammes dient, führt zu einem Unterprogramm MUSEIN. Hier werden die POKEY-Tonregister und die internen Variablen des Musikprogrammes gelöscht, anschließend wird der Programmteil MUSVBI in den 'Deferred' VBI eingefügt. Alles Weitere läuft jetzt bereits 'nebenbei' im VBI ab, MUSEIN beendet sich mit RTS und gibt damit die Kontrolle an das aufrufende Programm zurück.

Die VBI-Routine MUSVBI, die ab jetzt alle 1/50 sec. angestoßen wird, beinhaltet im wesentlichen eine Schleife, die für jeden Soundkanal einmal durchlaufen wird, also pro VBI viermal. In dieser Schleife werden jeweils zwei Unterprogramme, MUXPLY und MUXHUL, aufgerufen, die aktuelle Kanalnummer befindet sich dabei im X-Register.

MUXPLY ist dabei für die Abfolge der Noten verantwortlich, pro Aufruf bearbeitet es den im X-Register angegeben Kanal. Der Ablauf ist wie folgt:

4.) Nachprüfen, ob momentan schon eine Note auf diesem Kanal gespielt wird, wenn nein, dann zu Schritt 3.)

5.) Wird eine Note gespielt, so wird die Dauer der Note (MUDUR1-4) um eins vermindert. Ist die Dauer noch größer als Null, dann ist MUXPLY hier zu Ende.

6.) Kommt der Programmablauf an diese Stelle (MUXNEU), so ist die bisherige Note zu Ende, eine neue muss aus der Notentabelle geholt werden. Dazu wird die Basisadresse der Stimme <X> aus STMTBL/H besorgt, und mittels des Notenzeigers aus MUPNT1-4 indiziert in die Notentabelle zugegriffen. Ferner wird zu Beginn einer neuen Note der Hüllkurvenzeiger auf Null, d.h. auf Anfang der Hüllkurventabelle zurückgesetzt.

7.) Jetzt wird geprüft, ob es sich tatsächlich um einen Notenwert oder vielleicht um den Code des ENDE- oder START-Befehles handelt. War es 'START', so wird der Notenzeiger wieder auf Null gesetzt, die jeweilige Stimme beginnt damit wieder von vorne. Beim 'ENDE'-Befehl wird der Notenzeiger nicht weitergeschaltet, so dass beim nächsten Durchlauf wieder 'ENDE' erkannt wird, außerdem wird ein Pause-Befehl simuliert, dadurch bleibt die Stimme ruhig.

8.) War es ein ganz normaler Notenwert (einschließlich 'PAUSE'), dann wird seine Tondauer aus der Notentabelle gelesen, der Notenzeiger weitergeschaltet, und die Notenfrequenz an POKEY übergeben. Wurde ein 'PAUSE'-Befehl erkannt, wird ein Flag (MUPAU1-4) für den Hüllkurvengenerator gesetzt, so dass dieser Kanal ruhig bleibt.

Der Ablauf des Hüllkurvenunterprogrammes MUXHUL ist etwas einfacher:

1.) Wenn das Flag für PAUSE gesetzt ist, wird die Lautstärke des jeweiligen Kanals Null. Fertig!

2.) Sonst wird der nächste Lautstärkewert der Hüllkurve per Basisadresse (aus MHLTBL/H) und Hüllkurvenzeiger (MUHPT1-4) gelesen, mit Distortion-Bits für 'reinen Ton' versehen und an POKEY weitergegeben. Wurde aber der Wert 'ENDE' (der Hüllkurventabelle) erkannt, so wird POKEY nicht verändert und die Bearbeitung ist hier zu Ende. Das kann man sich zunutze machen, indem man eine kurze Hüllkurve mit einem Wert größer Null enden lässt, man erreicht so einen Anschlag, der in einen konstanten Ton übergeht und kann die Hüllkurventabellen relativ klein halten.

3.) Der Hüllkurvenzeiger wird auf den nächsten Tabellenwert weitergeschaltet.

MUXAUS: Hier wird der Musik-VBI wieder abgeschaltet und eventuell verbleibende Töne ausgeschaltet.

```
0100 ;*********************************************
0110 ;*                MUSIK IM VBI               *
0120 ;*                                           *
0130 ;* Musikprogramm mit Huellkurvengenerator    *
0140 ;*********************************************
0150 ;
0160 ;
0170 ;**************************************************
0180 ; Definition Noten/Frequenz und Notendauer
0190 ;**************************************************
0200 C3     =   243
0210 CIS3   =   230
0220 D3     =   217
0230 DIS3   =   204
0240 E3     =   193
0250 F3     =   182
0260 FIS3   =   173
0270 G3     =   162
0280 GIS3   =   153
0290 A3     =   144
0300 AIS3   =   136
0310 B3     =   128
0320 C4     =   121
0330 CIS4   =   114
0340 D4     =   108
0350 DIS4   =   102
0360 E4     =    96
0370 F4     =    91
0380 FIS4   =    85
0390 G4     =    81
0400 GIS4   =    76
0410 A4     =    72
0420 AIS4   =    68
0430 B4     =    64
0440 C5     =    60
0450 ;
0460 PAUSE  =   255        Code f. Pause
0470 P      =   PAUSE      Kuerzel f. Pause
0480 START  =   254        Code fuer Stimm-Neubeginn
0490 ENDE   =   253        Code fuer Stimmende
0500 ;
0510 DAUER  =   6          Dauer eines 16tel
0520 ;
0530 H      =   DAUER*8    Halbe Note
0540 Q      =   DAUER*4    Viertel Note (Quarter)
0550 E      =   DAUER*2    Achtel Note (Eight)
0560 S      =   DAUER      Sechzehntel Note
0570 ;
0580 ;**************************************************
0590 ;Systemkonstante, Hardware/O.S. Adressen
0600 ;**************************************************
0610 ;
0620 HI     =   $0100       Divisor fuer Hi-Byte
0630 LO     =   $FF         Maske fuer Lo-Byte
0640 ;
0650 SETVBV =   $E45C
0660 XITVBV =   $E462
0670 ;
0680 AUDF1  =   $D200
```

```
0690 AUDC1  =   $D201
0700 AUDCTL =   $D208
0710 SKCTL  =   $D20F
0720 ;
0730 ;**************************************************
0740 ; Interne Variablen fuer VBIMUSIK
0750 ;**************************************************
0760 ;
0770 MUDATA =   $9BE0      Variable in PAGE 6
0780 ;
0790 MUPNT1 =   MUDATA     Zeiger in Notentabelle
0800 MUDUR1 =   MUDATA+4   Zaehler fuer Tondauer
0810 MUPAU1 =   MUDATA+8   Flag fuer Pause
0820 MUHPT1 =   MUDATA+12  Zeiger in Huellkurve
0830 ;
0840 CSREGL =   $CE        Zeropageregister
0850 CSREGH =   $CF
0860 ;
0870 ;
0880 ;**************************************************
0890 ; Sprungleiste, wird von BASIC aufgerufen
0900 ; Einschalten : A=USR(38912)
0910 ; Ausschalten : A=USR(38916)
0920 ;**************************************************
0930 ;
0940        *=  $9800
0950        PLA            fuer BASIC-Aufruf per USR
0960        JMP MUSEIN     Musik einschalten
0970        PLA            fuer BASIC-Aufruf per USR
0980        JMP MUSAUS     Musik ausschalten
0990 ;
1000 ;**************************************************
1010 ; VBI-Routine, alle 1/50sec. aufgerufen
1020 ;**************************************************
1030 ;
1040 MUSVBI CLD            Dezimalmodus
1050        LDX #0         Kanal 0 zuerst spielen
1060 MUNXKN JSR MUXPLY     eine Stimme spielen >>
1070        JSR MUXHUL     Huellkurve f. eine Stimme >>
1080        INX            naechste Stimme
1090        CPX #4         schon alle 4 Stimmen?
1100        BNE MUNXKN     nein -->
1110 ;
1120        JMP XITVBV     dann VBI beenden ==>
1130 ;
1140 ;**************************************************
1150 ; Initialisierungsroutine
1160 ; POKEY init., Variable loeschen, VBI setzen
1170 ;**************************************************
1180 ;
1190 MUSEIN LDA #0         POKEY und Variablen loeschen
1200        LDX #3         je 4 Register
1210 ;
1220 MUSEN1 STA AUDF1,X    AUDF/C 1/2
1230        STA AUDF1+4,X  AUDF/C 3/4
1240        STA MUDUR1,X   Tondauer:=0
1250        STA MUPNT1,X   Notenzeiger:=Tabellenanfang
1260        STA MUHPT1,X   Huellk.-Zeiger:=Tabellenanfang
1270        DEX            schon alle 4?
1280        BPL MUSEN1     nein, nochmal -->
```

```
1290 ;
1300 ;
1310        LDA #3         Pokey ruecksetzen
1320        STA SKCTL
1330        LDA #0         Sound-Kontrollreg. zurueck
1340        STA AUDCTL
1350 ;
1360        LDY #MUSVBI&LO neuer VBI-Vektor LSB
1370        LDX #MUSVBI/HI MSB
1380        LDA #7         fuer Deferred VBI
1390        JSR SETVBV     O.S.-Routine, VBI setzen >>
1400        RTS
1410 ;
1420 ;**************************************************
1430 ; VBI ausschalten
1440 ; Vektor wird auf alten Wert zurueckgesetzt
1450 ;**************************************************
1460 ;
1470 MUSAUS LDY #XITVBV&LO alter VBI-Vektor, LSB
1480        LDX #XITVBV/HI MSB
1490        LDA #7         fuer Deferred VBI
1500        JSR SETVBV     >>>
1510 ;
1520        LDA #0         Sound aus!
1530        LDX #3         alle 4 Kanaele
1540 MUSAU1 STA AUDF1,X    AUDF/C 1/2
1550        STA AUDF1+4,X  AUDF/C 3/4
1560        DEX            schon alle 4?
1570        BPL MUSAU1     nein, nochmal -->
1580        RTS
1590 ;
1600 ;**************************************************
1610 ;Einen Kanal spielen
1620 ;pruefen der Tondauer, evtl. neue Note besorgen,
1630 ;Test auf ENDE, START, PAUSE
1640 ;**************************************************
1650 ;
1660 MUXPLY LDA MUDUR1,X   ist Kanal noch aktiv?
1670        BEQ MUXNEU     nein, dann neue Note
1680 ;
1690        DEC MUDUR1,X   sonst nur Tondauer - 1
1700        BNE MUXPEN     Ton laeuft noch --->
1710 ;
1720 MUXNEU LDA #0   Flag  fuer Pause vorerst
1730        STA MUPAU1,X   zuruecknehmen
1740        STA MUHPT1,X   Huellk. Zeiger auf Anf.
1750        LDA STMTBL,X   Basisadresse der Noten-
1760        STA CSREGL     tabelle in Zeropage-
1770        LDA STMTBH,X   register umspeichern
1780        STA CSREGH
1790 ;
1800        LDY MUPNT1,X   Zeiger in Notentabelle
1810        LDA (CSREGL),Y Tonhoehe der Note
1820        CMP #START     ist START-Befehl?
1830        BNE MUXN1      nein --->
1840 ;
1850        LDA #0         Notenzeiger auf Tabellen-
1860        STA MUPNT1,X   anfang richten
1870        JMP MUXNEU     und von Vorne ==>
1880 ;
```

```
1890 MUXN1 CMP #ENDE       ist ENDE-Befehl?
1900        BEQ MUXPAU     ja, dann Pause imitieren -->
1910 ;
1920        PHA            ;Tonhoehe merken
1930        INY            ;Dauer aus Notentabelle
1940        LDA (CSREGL),Y lesen
1950        STA MUDUR1,X   in Zaehler f. Notendauer
1960        INY            Noten Zeiger auf
1970        TYA            naechste Note richten
1980        STA MUPNT1,X   und abspeichern
1990 ;
2000        TXA            fuer POKEY Adressen
2010        ASL A          Index mal 2
2020        TAY
2030        PLA            ;Tonhoehe wieder holen
2040        CMP #PAUSE     war es ein PAUSE Befehl?
2050        BEQ MUXPAU     ja ,dann Pausenbearbeitung
2060 ;
2070        STA AUDF1,Y    sonst Note an POKEY
2080 MUXPEN RTS
2090 ;
2100 MUXPAU LDA #PAUSE     Huellk. ausschalten
2110        STA MUPAU1,X
2120        RTS
2130 ;
2140 ;**************************************************
2150 ; Huellkurvengenerator
2160 ; Lautstaerke fuer einen Kanal wird gemaess
2170 ; Huellkurve eingestellt
2180 ;**************************************************
2190 ;
2200 MUXHUL LDA MUPAU1,X   Flag fuer Pause?
2210        CMP #PAUSE     ist auf Pause?
2220        BNE MUXH1      nein, normale Huellk.
2230 ;
2240        LDA #0         Pause: Lautstaerke auf Null
2250        PHA            Lautst. merken
2260        BEQ MUXH2      (immer!) an Pokey -->
2270 ;
2280 MUXH1 LDA MHLTBL,X    Huellkurven Basisadresse
2290        STA CSREGL     in Zeropage Register
2300        LDA MHLTBH,X   MSB
2310        STA CSREGH
2320        LDY MUHPT1,X   aktueller Zeiger in Hk.
2330        LDA (CSREGL),Y neuer Stuetzpunkt
2340        CMP #ENDE      ist Huellk. zu Ende?
2350        BEQ MUXHEN     ja, dann fertig!
2360 ;
2370        PHA            ;Stuetzpunkt merken
2380        INY            ;HK-Zeiger weiterschalten
2390        TYA            und abspeichern
2400        STA MUHPT1,X
2410 ;
2420 MUXH2 TXA             ;Offset fuer Pokey-Register
2430        ASL A          Index mal 2
2440        TAY
2450        PLA            ;Hk-Stuetzpunkt
2460        ORA #10*16     POKEY-Modus: reiner Ton
2470        STA AUDC1,Y    an POKEY geben
2480 ;
```

```
2490 MUXHEN RTS
2500 ;
2510 ;**************************************************
2520 ;Tabellen der Basisadressen fuer Noten-
2530 ;und Huellkurven Tabellen.
2540 ;**************************************************
2550 ;
2560 STMTBL .BYTE STIMM1&LO,STIMM2&LO,STIMM3&LO,STIMM4&LO
2570 STMTBH .BYTE STIMM1/HI,STIMM2/HI,STIMM3/HI,STIMM4/HI
2580 ;
2590 ;
2600 MHLTBL .BYTE HUELL1&LO,HUELL2&LO,HUELL3&LO,HUELL4&LO
2610 MHLTBH .BYTE HUELL1/HI,HUELL2/HI,HUELL3/HI,HUELL4/HI
2620 ;
2630 ;**************************************************
2640 ;Notentabellen:
2650 ;jede Tabelle beginnt mit Label STIMM<n>
2660 ;wobei <n>:Nummer der Stimme (1-4)
2670 ;danach folgen die Noten (Note,Dauer)
2680 ;jeweils in .BYTE Befehlen.
2690 ;nach letzter Note muss 'ENDE' oder
2700 ;'START' eingetragen werden.
2710 ;**************************************************
2720 ;
2730 STIMM1
2740 ;
2750        .BYTE E4,Q,E4,Q,D4,Q,D4,E,E4,E
2760        .BYTE F4,Q,F4,Q,E4,Q,E4,Q
2770        .BYTE E4,E,D4,E,E4,E,F4,E,G4,Q,F4,E,E4,E
2780        .BYTE D4,Q+E,C4,E,C4,H
2790        .BYTE E4,Q,E4,Q,F4,Q,F4,E,E4,E
2800        .BYTE D4,Q,D4,Q,E4,Q,E4,E,D4,E
2810        .BYTE C4,E,B3,E,C4,E,D4,E,E4,Q,D4,E,C4,E
2820        .BYTE B3,Q+E,A3,E,A3,H
2830        .BYTE G4,Q,G4,Q,A4,Q,F4,E,E4,E
2840        .BYTE D4,Q,D4,Q,G4,Q,F4,E,D4,E
2850        .BYTE C4,E,D4,E,E4,E,F4,E,G4,Q,F4,E,E4,E
2860        .BYTE D4,Q+E,C4,E,C4,H
2870        .BYTE ENDE
2880 ;
2890 STIMM2
2900 ;
2910        .BYTE C4,Q,C4,Q,B3,Q,B3,E,C4,E
2920        .BYTE D4,Q,D4,Q,C4,Q,C4,Q
2930        .BYTE C4,E,B3,E,C4,E,D4,E,E4,Q,D4,E,C4,E
2940        .BYTE C4,Q,B3,Q,C4,H
2950        .BYTE C4,Q,C4,Q,D4,Q,D4,E,C4,E
2960        .BYTE B3,Q,B3,Q,C4,Q,C4,E,B3,E
2970        .BYTE A3,E,GIS3,E,A3,E,B3,E,C4,Q,B3,E,A3,E
2980        .BYTE A3,Q,GIS3,Q,A3,H
2990        .BYTE E4,Q,E4,Q,F4,Q,D4,E,C4,E
3000        .BYTE B3,Q,B3,Q,E4,Q,C4,E,B3,E
3010        .BYTE A3,E,B3,E,C4,E,D4,E,E4,Q,D4,E,C4,E
3020        .BYTE C4,Q,B3,Q,C4,H
3030        .BYTE ENDE
3040 ;
3050 STIMM3
3060 ;
3070        .BYTE C3,E,D3,E,E3,E,F3,E,G3,Q,G3,Q
3080        .BYTE D3,E,E3,E,F3,E,G3,E,A3,Q,A3,Q
```

```
3090        .BYTE G3,Q,F3,Q,E3,Q,F3,Q
3100        .BYTE G3,Q,G3,Q,C3,H
3110        .BYTE A3,E,G3,E,F3,E,E3,E,D3,Q,D3,Q
3120        .BYTE G3,E,F3,E,E3,E,D3,E,C3,Q,C3,Q
3130        .BYTE F3,Q,F3,Q,E3,Q,A3,Q
3140        .BYTE D3,Q,E3,Q,A3,H
3150        .BYTE C4,E,B3,E,A3,E,G3,E,F3,Q,F3,Q
3160        .BYTE G3,E,F3,E,E3,E,D3,E,C3,Q,C3,Q
3170        .BYTE A3,Q,G3,E,F3,E,E3,E,C3,E,D3,E,E3,E
3180        .BYTE F3,Q,G3,Q,C3,H
3190        .BYTE ENDE
3200 ;
3210 ;
3220 ;
3230 STIMM4
3240 ;
3250        .BYTE ENDE
3260 ; wird im Beispiel nicht benutzt
3270 ;
3280 ;**************************************************
3290 ; HUELLKURVENTABELLEN
3300 ; Hier werden Stuetzpunkte der Huellkurven
3310 ; als Lautstaerkewerte eingetragen
3320 ; jede Tabelle beginnt mit HUELL<n>
3330 ; und endet mit 'ENDE'
3340 ;**************************************************
3350 ;
3360 HUELL1 .BYTE 10,10,8,7,6,9,8,8,8,7,7,7
3370        .BYTE 6,6,6,5,5,4,3,2,1,1,0,ENDE
3380 HUELL2 .BYTE 2,6,8,8,8,7,7,7,6,6,6,6
3390        .BYTE 5,5,4,4,4,3,2,2,2,1,0,ENDE
3400 ;
3410 HUELL3 .BYTE 2,4,6,8,9,10,9,8,8,7,7,6
3420        .BYTE 6,6,6,6,5,5,5,5,5,5,5,5,5,5
3430        .BYTE 4,4,4,4,4,4,4,4,4,4,2,1,0,ENDE
3440 HUELL4 .BYTE ENDE
3450 ;
3460        .OPT NO LIST
```

Verwendung in eigenen Programmen

Das Musikprogramm an sich ist schon eine tolle Sache, nur wie bekommen Sie die Musik in Ihre eigenen Programme? Dazu gibt es gleich mehrere Möglichkeiten:

1.) Wandeln Sie das assemblierte .OBJ File in einen BASIC-Loader um, den Sie dann per 'ENTER' in Ihr Programm übernehmen können. Das Programm DATGEN **am** Schluss dieses Buches hilft Ihnen dabei. Die Nachteile dieser Methode wurden bereits besprochen, der Vorteil ist, dass Sie ein einziges in sich geschlossenes Programm bekommen.

2.) Einlesen des Maschinenprogrammes von der Diskette. Dazu können Sie das Hilfsprogramm zum Laden von Binary-Load Files im Abschnitt 'BLS...' verwenden. **Nachteil:** Ihr Programm besteht dann aus mehreren Files!

3.) Verwendung eines EXECUTE-Files unter OS/A+ DOS. Das Maschinenprogramm wird Execute-gesteuert mit einem DOS-'LOAD' Befehl eingelesen. Diese Methode eignet sich hervorragend für die Entwicklungsphase eines Programmes. Nähere Erklärungen finden Sie in den Abschnitten 'Batch-Verarbeitung und 'BCOM...'. Das STARTUP.EXC auf der HK-Programmdiskette arbeitet auch nach diesem Prinzip.

Bei allen drei Methoden muss man natürlich dafür Sorge tragen, dass der Speicherbereich ab $9800 vor dem Zugriff anderer Programme geschützt ist, am leichtesten erreicht man das mit

POKE 106,144;GR,0;REM 4K reservieren!

Lesen Sie dazu auch den Abschnitt 'Wohin mit Maschinenprogrammen'.

Einige Anregungen

Mit diesem Programm können Sie Ihrem ATARI schon recht hübsche Musik entlocken, fühlen Sie sich frei das in eigenen Programmen zu verwenden. Jedes Programm gewinnt bei der Verwendung von guten akustischen Mitteln garantiert an Reiz. Es lohnt sich bestimmt, das Programm nach eigenen Ideen zu erweitern und zu verändern, Ihr Computer ist damit längst noch nicht am Ende seiner Leistungsfähigkeit! Einige Vorschläge:

Verkürzen der Hüllkurventabelle durch sogenannte - ADSR-Darstellung (Attack-Decay-Sustain-Release). Die Hüllkurve ließe sich damit mit max. 4 Bytes abspeichern.

- Einbau eines sog. SLIDE-Effektes. Die Frequenz wird beim Übergang zwischen zwei Noten nicht mehr abrupt geändert, sondern geht fließend ineinander über.
- Einbau eines Vibrato-Effektes.
- Erweiterung auf Bassnoten, die mittels des Soundmodus 12 (4-Bit Polycounter) erzeugt werden können.
- ...

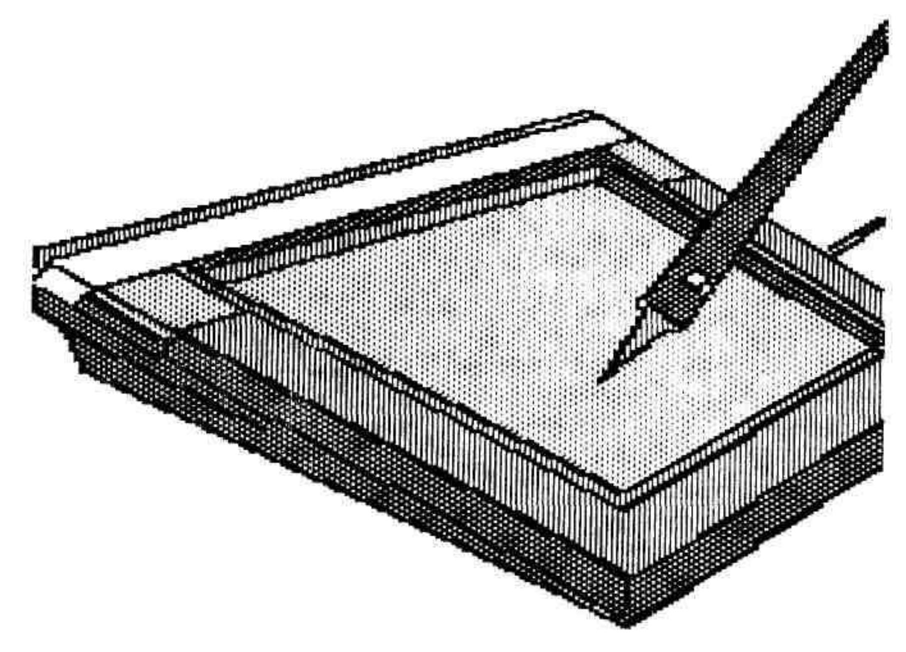
Sie sehen, die Möglichkeiten sind ungeheuer vielfältig. wenn Sie eine der oben genannten Ideen inspiriert hat, so wünsche ich Ihnen viel Spaß bei der Programmierung Ihres eigenen MUSIKGENERATORS.

```
31000 REM * BASIC-Loader fuer VBI-Musikprogramm
31020 POKE 106,144:GRAPHICS 0:POSITION 5,10:? "Initialisierung ..."
31030 GOSUB 32000
31040 GRAPHICS 0:? "Bitte mit A=USR(MUAUS) abschalten!"
31050 A=USR(MUEIN)
31090 END
32000 REM * Binaer-File laden
32010 S=0:RESTORE 32100
32020 FOR A=38912 TO 39592:READ D:POKE A,D:S=S+D:NEXT A
32030 IF S<>63952 THEN ? "DATEN-FEHLER!":STOP
32040 MUEIN=38912:MUAUS=MUEIN+4
32090 RETURN
32100 DATA 104,76,25,152,104,76,67,152,216,162,0,32,90,152,32,169
32110 DATA 152,232,224,4,208,245,76,98,228,169,0,162,3,157,0,210,157
32120 DATA 4,210,157,228,155,157,224,155,157,236,155,202,16,238,169
32130 DATA 3,141,15,210,169,0,141,8,210,160,8,162,152,169,7,32,92
32140 DATA 228,96,160,98,162,228,169,7,32,92,228,169,0,162,3,157,0
32150 DATA 210,157,4,210,202,16,247,96,189,228,155,240,5,222,228,155
32160 DATA 208,62,169,0,157,232,155,157,236,155,189,216,152,133,206
32170 DATA 189,220,152,133,207,188,224,155,177,206,201,254,208,8,169
32180 DATA 0,157,224,155,76,100,152,201,253,240,24,72,200,177,206
32190 DATA 157,228,155,200,152,157,224,155,138,10,168,104,201,255
32200 DATA 240,4,153,0,210,96,169,255,157,232,155,96,189,232,155,201
32210 DATA 255,208,5,169,0,72,240,25,189,224,152,133,206,189,228,152
32220 DATA 133,207,188,236,155,177,206,201,253,240,15,72,200,152,157
32230 DATA 236,155,138,10,168,104,9,160,153,1,210,96,232,95,214,79
32240 DATA 152,153,153,154,80,104,128,168,154,154,154,154,96,24,96
32250 DATA 24,108,24,108,12,96,12,91,24,91,24,96,24,96,24,96,12,108
32260 DATA 12,96,12,91,12,81,24,91,12,96,12,108,36,121,12,121,48,96
32270 DATA 24,96,24,91,24,91,12,96,12,108,24,108,24,96,24,96,12,108
32280 DATA 12,121,12,128,12,121,12,108,12,96,24,108,12,121,12,128
32290 DATA 36,144,12,144,48,81,24,81,24,72,24,91,12,96,12,108,24,108
32300 DATA 24,81,24,91,12,108,12,121,12,108,12,96,12,91,12,81,24,91
32310 DATA 12,96,12,108,36,121,12,121,48,253,121,24,121,24,128,24
32320 DATA 128,12,121,12,108,24,108,24,121,24,121,24,121,12,128,12
32330 DATA 121,12,108,12,96,24,108,12,121,12,121,24,128,24,121,48
32340 DATA 121,24,121,24,108,24,108,12,121,12,128,24,128,24,121,24
32350 DATA 121,12,128,12,144,12,153,12,144,12,128,12,121,24,128,12
32360 DATA 144,12,144,24,153,24,144,48,96,24,96,24,91,24,108,12,121
32370 DATA 12,128,24,128,24,96,24,121,12,128,12,144,12,128,12,121
32380 DATA 12,108,12,96,24,108,12,121,12,121,24,128,24,121,48,253
32390 DATA 243,12,217,12,193,12,182,12,162,24,162,24,217,12,193,12
32400 DATA 182,12,162,12,144,24,144,24,162,24,182,24,193,24,182,24
32410 DATA 162,24,162,24,243,48,144,12,162,12,182,12,193,12,217,24
32420 DATA 217,24,162,12,182,12,193,12,217,12,243,24,243,24,182,24
32430 DATA 182,24,193,24,144,24,217,24,193,24,144,48,121,12,128,12
32440 DATA 144,12,162,12,182,24,182,24,162,12,182,12,193,12,217,12
32450 DATA 243,24,243,24,144,24,162,12,182,12,193,12,243,12,217,12
32460 DATA 193,12,182,24,162,24,243,48,253,253,10,10,8,7,6,9,8,8,8
32470 DATA 7,7,7,6,6,6,5,5,4,3,2,1,1,0,253,2,6,8,8,8,7,7,7,6,6,6,6
32480 DATA 5,5,4,4,4,3,2,2,2,1,0,253,2,4,6,8,9,10,9,8,8,7,7,6,6,6
32490 DATA 6,6,5,5,5,5,5,5,5,5,5,5,4,4,4,4,4,4,4,4,4,4,2,1,0,253,253
```

BASIC-Loader für Musikprogramm. HK-Disk:MUSIK.BAS

Das ATARI TOUCH TABLET

Das Touch Tablet (oder auch 'Maltafel') ist eines der universellsten und einfachst zu bedienenden Eingabegeräte für Atari-Computer. Endlich ist Schluss mit zittriger Joystick Malerei, die trotz hervorragender Graphikprogramme nicht gerade das Gelbe vom Ei ist. Aber, das Touch Tablet lässt sich zu viel, viel mehr gebrauchen! Seine überragenden Fähigkeiten können auch in eigenen Programmen äußerst gewinnbringend eingesetzt werden.



Atari Touch Tablet

Programmierung des Touch Tablets

Mit dem Touch Tablet haben Sie zwei Eingabemöglichkeiten:

a) die berührungsempfindliche Fläche, die wie zwei Paddles gelesen wird.

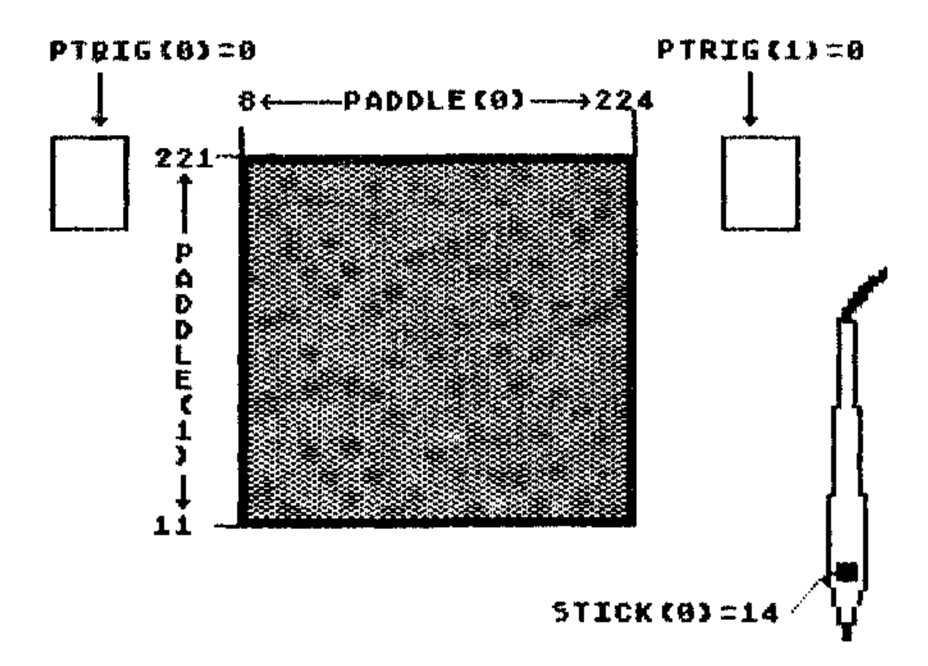
   X=PADDLE(0) liest Abstand vom linken Rand
   Y=PADDLE(1) liest Abstand vom unteren Rand

   Wird die Maltafel nicht berührt, so nehmen X und Y den Wert *228* an.

   Kompatibilität zum Koala-Pad: Leider verhalten sich das Atari Tablet und das Koala-Pad in der vertikalen Richtung gerade spiegelverkehrt, so dass beide softwaremäßig nicht kompatibel sind. Schade!

b) Weiterhin sind drei Taster vorhanden, die wie ein Joystick (oder Paddle-Trigger) abgefragt werden können.

   STICK(0) = 15: kein Taster betätigt
   = 14: Taster am Stift gedrückt
   = 11: linker Taster, oder: PTRIG(0)=0
   = 7: rechter Taster, oder: PTRIB(1)=0



Touch Tablet Funktionen

Hier zwei kleine (aber interessante) Spielereien, die Ihnen zeigen, wie die Maltafel programmiert werden kann.

* Tablet-Sound:

```
10 P=PADDLE(0):IF P=22B THEN SOUND 0,0,0,0:GOTO 10
20 SOUND 0,P,10,8 :GOTO 10
```

* Farbeinstellung mit Maltafel:

```
10 P0=PADDLE(0):P1=PADDLE(1):IF P0=228 OR PI=228 THEN 10
20 SETCOLOR 2,P0/15,P1/15:GOTO 10
```

Positionierung eines Cursors am Bildschirm

Die erste und einfachere Möglichkeit, die Position eines Objektes am Bildschirm mit dem Touch Tablet festzulegen, ist die absolute Positionierung. Dabei wird ganz simpel die X-und Y-Koordinate des Cursors direkt aus den Paddlewerten des Tablets ermittelt.

Ein Beispiel in BASIC:

```
10 REM ** Absolute Cursor-Positionierung mit Touch Tablet
30 OPEN #3,4,0,"K:"
100 IF PEEK(764)<>255 THEN GET #3,A:? CHR$(A);:REM * Taste gedrueckt?
110 XP=PADDLE(0):YP=PADDLE(1):IF XP=228 OR YP=228 THEN 100
120 GOSUB 10000
190 GOTO 100
10000 REM * Berechnung der Cursorposition
10010 XC=(XP-10)/5:YC=23-(YP-20)/8
10020 IF YC>23 THEN YC=23:REM Grenzen pruefen
10030 IF YC<0 THEN YC=0
10040 IF XC>39 THEN XC=39
10050 IF XC<2 THEN XC=2
10060 POKE 84,YC:POKE 85,XC:? CHR$(31);CHR$(30);:REM Cursor fuehren
10090 RETURN
```

Absolute Positionierung mit Touch Tablet

Die zweite Methode, hier als relative Positionierung bezeichnet, ist etwas komplizierter, Die Koordinaten werden nicht wie im obigen Beispiel unmittelbar aus der Stiftposition am Tablet berechnet, sondern beziehen sich immer auf den letzten Aufsetzpunkt, Tippen Sie das unten abgedruckte Beispiel gleich ein (es befindet sich auch auf der HK-Disk: TOUCH2,BAS) und probieren Sie es aus. Wenn Sie den Stift (oder Finger!) an beliebiger Stelle der Maltafel aufsetzen, bleibt der Cursor noch an seiner Stelle. Erst wenn Sie eine Bewegung vornehmem, folgt der Cursor nach. Haben Sie das Ende des Tablets erreicht, so heben Sie den Stift ab, setzen ihn am gegenüberliegenden Ende wieder auf und können dann den Cursor weiterschieben,

```
10 REM ** Relative Cursor-Positionierung mit Touch Tablet
20 YRANGE=16:XRANGE=16:REM * Empfindlichkeit
30 OPEN #3,4,0,"K:"
100 IF PEEK(764)<>255 THEN GET #3,A:? CHR$(A);
110 XP=PADDLE(0):YP=PADDLE(1):IF XP=228 OR YP=228 THEN 100
120 GOSUB 10000:REM Tablet beruehrt!
190 GOTO 100
10000 REM * Cursorsteuerung
10010 XC=PEEK(85):YC=PEEK(84):XAUF=PADDLE(0):YAUF=PADDLE(1)
10020 XP=PADDLE(0):YP=PADDLE(1):REM * neue Position
10030 IF XP=228 OR YP=228 THEN RETURN :REM * abgehoben
10035 XP=(XP+XAUF)/2:YP=(YP+YAUF)/2:REM * Mittelwert
10040 XC=XC+(XP-XAUF)/XRANGE:YC=YC-(YP-YAUF)/YRANGE:XAUF=XP:YAUF=YP
10050 IF YC>23 THEN YC=23:REM Grenzen pruefen
10060 IF YC<0 THEN YC=0
10070 IF XC>39 THEN XC=39
10080 IF XC<2 THEN XC=2
10090 POKE 84,YC:POKE 85,XC:? CHR$(31);CHR$(30);:REM Cursor fuehren
10100 GOTO 10020
```

Relative Positionierung mit Touch Tablet

Vorzüge dieses Verfahrens: Das leichte Zittern des Cursors vom ersten Beispiel wird vermieden, außerdem ist es mit dieser Methode möglich, Graphiken zu editieren, die die Auflösung des Tablets überschreiten. Zusätzlich wird im Beispielprogramm der relativen Positionierung eine Mittelwertbildung von jeweils zwei aufeinander folgenden Werten vorgenommen, die zur weiteren Beruhigung des Cursors beiträgt.

Zum Schluss noch ein Tipp zur AtariArtist-Cartridge

Wie Sie sicherlich wissen, werden Bilder in einem komprimierten Format aufgezeichnet, so dass sie zum Teil erheblich weniger Speicherplatz als die sonst obligatorischen 62 Sektoren benötigen. Das ist recht schön, wenn man viele Bilder auf einer Diskette aufbewahren möchte, aber weniger angenehm, wenn man die Bilder in eigenen Programmen verwenden möchte.

Zum Glück gibt es noch einen Trick, der nicht in der Bedienungsanleitung steht: Mit der Taste INSERT können Sie das momentane Bild als ganz normales Bildfile unter dem fest vorgegebenen Namen PICTURE aufzeichnen, mit CLEAR ist es möglich ein Bild des gleichen Filenamens einzulesen. damit ist es leicht möglich, Bilder aus anderen oder in andere Graphikprogramme zu übernehmen oder auch mit dem Bild-Loader aus diesem Buch zu bearbeiten. Allerdings geht mit dieser Methode die Farbregisterinformation verloren, es ist aber kein Problem, die Farben wieder von Hand zu ergänzen.

Soundgenerator im VBI mit komfortablem Editor

In vielen BASIC—Programmen kann man folgendes Problem beobachten; Während ein Soundeffekt erzeugt wird, erstarren alle Spielfiguren in ihrer Bewegung, da BASIC einfach zu langsam ist, um interessante Toneffekte gleichzeitig mit Bildschirmbewegungen ablaufen zu lassen. In Maschinensprache ist das natürlich möglich, wenn es auch nicht unbedingt einfach ist. da hier leider kein komfortabler 'SOUND'-Befehl zur Verfügung steht. Mit dem SOUND—GENERATOR werden diese Mängel behoben: Nun können Sie in BASIC ganze Abfolgen von in sich dynamischen Sounds mit einem Befehl aufrufen, die dann unabhängig von BASIC laufen. Auch wenn Sie in Maschinensprache programmieren, ist das in den beiden nachfolgenden Abschnitten beschriebene Soundsystem extrem nützlich, da die Töne mittels eines komfortablen Editors zusammengestellt werden.

Features.

- Töne werden interaktiv mit Editor erstellt,
- bis zu 32 Töne werden abgespeichert
- Verkettung von Sounds möglich
- Gleichzeitiger Aufruf von mehreren Sounds möglich
- Verwendung in BASIC und Assembler
- Computer steht während der Sounderzeugung voll zur Verfügung

Wie funktioniert 's?

Mit Hilfe des SOUND-EDITOR Programmes können Sie bis zu 32 verschiedene Sounds zusammenstellen und diese sofort originalgetreu hören. Wenn Sie diese Töne in Ihren eigenen Programmen verwenden wollen, können Sie diese auf Diskette abspeichern und mittels eines kleinen Zusatzprogrammes SNDLOAD.BAS in Ihrem Programm unterbringen. So werden Änderungen der Sounds zur Leichtigkeit: Wenn Sie feststellen, dass der eine oder andere Sound doch nicht so recht zu Ihrem Programm passt, so können Sie die Sound-Datei wieder mit dem Editor laden, die Töne neu editieren und wieder abspeichern. Beim nächsten Lauf Ihres Programmes sind dann die neuen Sounds schon enthalten.

DER SOUNDEDITOR

Auf den nächsten Seiten finden Sie ein Listing des Editorprogrammes. Sie können das Programm eintippen, oder, falls Sie die 'Hexenküche' Diskette haben, das Programm 'SNDEDIT.BAS' laden:

```
RUN"D:SNDEDIT.BAS"
```

Gehören Sie zu denjenigen, die das Programm eintippen müssen, so sei Ihnen folgendes mit auf den Weg gegeben: Achten Sie besonders auf die DATA-Befehle am Ende des Programmes. Speichern Sie das Programm in jedem Fall ab, bevor Sie es zum ersten Mal starten. Meldet das Programm einen 'DATEN-FEHLER', so haben Sie sich bei den DATAs vertippt, es empfiehlt sich dann ein nochmaliger Vergleich mit dem Listinq.

Wenn Sie alles richtig gemacht haben, so muss sich der S0UND-EDIT0R mit folgendem Menü melden:

```
SOUNDEDITOR                  P. Finzel
        Musternummer             : 0
        Freq. Anfang             : 0
        Freq. Aenderung          : 0
        Laut. Anfang             : 0
        Laut. Aenderung          : 0
        Modus                    : 5 &17 Bit Poly
        Laenge                   : 0
        Kanal Nr.                : 0
        Prioritaet               : 0
        Naechst. Muster          : NIL

          TRIGGER FUER SOUND
  START: LOAD FILE    OPTION: SAVE FILE
```

Die Bedienung des Soundeditors ist denkbar einfach: Sie stecken einen Steuerknüppel in Port 1 und schon geht's los. Bewegung nach oben oder unten wählt die Zeile, Bewegung nach links bzw. rechts verkleinert bzw. vergrößert den Wert dieser Zeile. Ausnahme bildet hier nur die Zeile 'Modus', hier wählen Sie mit der Joystickbewegung nach links/rechts die Art der Geräuscherzeugung aus.

Was können Sie jetzt alles einstellen?

Musternummer: (0 - 31)
Hier können Sie einen Ton von 32 möglichen auswählen, der im Moment editieren werden soll.

Freq. Anfang: (0 - 255)
Tonhöhe, die im Moment des Anstoßes eingestellt wird. Dabei ist '0' der höchste Ton, '255' der tiefste.

Freq. Aenderung: (-127 - +127)
Als Frequenzänderung wird der Wert bezeichnet, um den die Tonhöhe pro 1/50 sec. verändert wird. Negative Werte erhöhen, positive Werte erniedrigen den Ton. Versuchen Sie es hier einmal mit relativ großen Werten (z.B. 50 oder -60), das ergibt interessante 'blubbernde' Töne.

Laut. Anfang: (0-255)
Lautstärke im Moment des Anstoßes. Nanu? Lautstärke kann man doch normalerweise nur in 16 Stufen (0-15) einstellen! Richtig, das ist auch hier so, aber um die Auflösung der Lautstärkeänderung zu erhöhen wird die Lautstärke multipliziert mit 16 eingetragen. Lautstärke Anfang 128 ist also gleich Lautstärke 8 des SOUND-Befehles.

Laut. Aenderung: (-127 - +127)
Änderung der Lautstärke pro 1/50 sec. Wie oben ist dieser Wert wieder multipliziert mit 16 zu denken, so wird eine feinere Unterteilung erreicht. Wenn Sie den Wert 1 einstellen, so wird die Lautstärke erst alle 16 Takte (zu je 1/50 sec) um eine Stufe erhöht. Interessant sind auch hier wieder relativ große Werte (z.B. 40), die zu abgehackten, sich wiederholenden Geräuschen führen.

Modus

Damit können Sie den POKEY—Tonerzeugungsmodus einschalten. Das entspricht in etwa dem Parameter 'Distortion' beim BASIC SOUND-Befehl. Folgende Möglichkeiten sind einstellbar:

5 & 17 Bit Poly	:	'unreines' Rauschen
5 Bit Poly	:	monotone Verzerrung
5 & 4 Bit Poly	:	motorenähnliches Geräusch
17 Bit Poly	:	weißes Rauschen
Purer Ton	:	reiner Ton, Rechteckschwingung
4 Bit Poly	:	verzerrter Ton

Laenge: (0-255)

Länge des Tones in 1/50 sec.

Kanal Nr.: (0 — 3)

Vier Hardware-Tongeneratoren stehen zur Verfügung.

Priorität: (0 — 32)

Hiermit hat es eine besondere Bewandtnis: Obwohl der Atari dem Programmierer vier Tonkanäle zur Verfügung stellt, kann es vorkommen, dass ein momentan noch belegter Kanal die Aufforderung zum Beginn eines neuen Tones erhält, dabei ist besonders an Spiele mit viel Action zu denken. Um derartigen Situationen gerecht zu werden, kann man jedem Sound eine 'Priorität' zuordnen, die über die Wichtigkeit eines Tones entscheidet. Wird ein Ton niedriger Priorität (z.B. 0) gespielt, und dabei ein Ton mit höherer Priorität (z.B. 3) angefordert, so wird der niederpriore Ton abgeschaltet und der neue Ton begonnen. Umgekehrt kann aber ein Ton mit niedriger Priorität einen Ton mit höherer nicht unterbrechen.

Naechst. Muster: (0 — 31)

Mittels dieser Option können Sie sehr komplexe Soundabfolgen einstellen, indem Sie einfach mehrere Töne miteinander verketten. Normalerweise ist hier 'NIL' eingestellt, dies soll das Ende einer Tonliste veranschaulichen. Wenn Sie hier die Nummer eines Tones (von 0-31) einstellen, dann wird dieser nach Ablauf des momentanen Tones aufgerufen. Der zweite Ton kann natürlich wieder einen Verweis auf einen neuen Sound beinhalten usw... Auf diese Weise ist es sogar möglich kleine Melodien zu programmieren. Ein weitere interessante Anwendung ist die Verzweigung auf sich selbst. Das eignet sich hervorragend zum Editieren eines Tones, da man jede Änderung sofort hören kann. Abgeschaltet wird so ein Ton entweder durch Änderung des Musterverweises auf 'NIL' (im Editor) oder durch Aufruf eines höherprioren Musters im selben Kanal.

BEISPIEL — BEISPIEL — BEISPIEL —
Lassen Sie die Musternummer auf 0, stellen Sie den Frequenzanfang auf 30, Lautstärkeanfang auf 60. Wählen Sie den Modus 'Purer Ton' und die Länge 20. Auf Knopfdruck hören Sie jetzt einen einfachen reinen Ton. Experimentieren Sie jetzt mit der Frequenzänderung und der Lautstärkeänderung. Sie werden staunen, welch interessante Töne Sie Ihrem Computer entlocken können. Ein ungeheureres Potential für eigene Tonkreationen stellen die Modi '4—Bit Poly' und '5-Bit Poly' dar, besonders in höheren Tonlagen. Wenn Sie einen Ton herausgefunden haben, den Sie bewahren möchten, brauchen Sie nur eine andere Musternummer zu wählen.

ABSPEICHERN DER SOUND-DATEI
Jeweils ein Satz von 32 Tönen kann komplett auf Disk geschrieben werden. Dazu drücken Sie die OPTION Taste und geben dann den Filenamen im Standard Atari Format ein:

D:FILENAME.EXT

Empfehlenswert ist der Extender .SND, an dem man später die Sound-Datei sofort erkennen kann.

Beispiel: *OPTION* drücken, 'D:TEST.SND' eingaben, *RETURN* drücken, und die Sound-Datei wird auf Disk geschrieben.

Wie schon erwähnt, können Sie eine Sound—Datei jederzeit wieder einlesen und editieren. Dazu drücken Sie *START* und geben den Filenamen der zu editierenden Datei ein.

Programmübersicht: SOUND-EDITOR
Zeile

100—210	:	Aufruf der Initialisierungs—Programme
200-390	:	Hauptschleife, wird endlos durchlaufen
1000-1190	:	Gibt Bildschirmmaske aus. Kursive Schrift bedeutet Reverse-Zeichen!
2000-2090	:	Hier wird das momentan zu bearbeitende Feld ausgewählt, zusätzlich wird bei gedrücktem Knopf der aktuelle Sound ausgelöst.
3000-3090	:	Unterprogramme zum Verändern der einzelnen Felder. R gibt relative Position in der Soundtabelle an, MN und MX bestimmen den minimalen bzw. maximalen Wert des Feldes, H die kleinste Schrittweite der Veränderung.
4000-4970	:	Unterprogramme zum Ausdrucken des jeweiligen Feldes. Zeile 4020 übernimmt die Verteilung.
5000-5090	:	Druckt den Wert aller Felder aus.
6000-6090	:	SAVE-Routine zum Aufzeichnen der Sound-Datei
7000-7090	:	LOAD-Routine, einlesen der Datei.
7500-7590	:	Filenamen einlesen.
3000-8710	:	Sammlung von Hilfsroutinen zum Verändern der Felder.
9000-9320	:	Hilfsroutinen zum Löschen der Soundtabelle, Meldungen drucken, VBI ...
10000-10010	:	Bezeichnungen für POKEY-Modi
72000-Ende	:	Soundmaschinenprogramm in DATA's

```
10 REM ******************************
20 REM *         SOUND EDITOR       *
30 REM *        fuer VBI-Sound      *
40 REM ******************************
50 REM
100 POKE 106,144:GRAPHICS 2+16:POSITION 0,5:? #6;"EINEN MOMENT BIT-
TE!"
110 CONSOL=53279
120 DIM D$(40),F$(20)
130 GOSUB 32000:REM * Maschinenprogramm 'poken'
140 GOSUB 9000:REM * Soundtabelle loeschen
150 GOSUB 9300:REM * VBI einschalten
200 GOSUB 1000:POKE 559,34:GOSUB 9100
210 MUSTER=0:INDEX=SNDTAB:N=1:C=19:GOSUB 5000
300 REM * Hauptschleife des ProgrammeS ************
310 GOSUB 2000:REM * Feld waehlen
320 GOSUB 3000:REM * Aenderung durchfuehren
330 GOSUB 4000:REM * Feld ausdrucken
390 GOTO 300
400 REM ******************************************
1000 REM *** Bildschirmmaske drucken ***
1005 GRAPHICS 0:POKE 752,1:SETCOLOR 1,0,8:SETCOLOR 2,11,0
1010 ? "     SOUND-EDITOR       P.FINZEL 1984":REM Kursiv bed. 're-
verse'
1020 ? :? :? "Muster Nr.     :"
1030 ? :? "Freq. Anfang   :"
1040 ? "Freq. Aenderung:"
1050 ? :? "Laut. Anfang   :"
1060 ? "Laut. Aenderung:"
1070 ? :? "Modus          :"
1080 ? "Laenge         :"
1090 ? "Kanal Nr.      :"
1100 ? "Priotitaet     :"
1110 ? :? "Naechst. Muster:"
1180 POSITION 2,22:? "START: LOAD FILE - OPTION: SAVE FILE"
1190 RETURN
2000 REM *** Auswahl der Felder ***
2010 ST=STICK(0):POKE 77,0:IF ST=13 OR ST=14 THEN GOSUB 9200
2020 N=N+(ST=13 AND N<10)-(ST=14 AND N>1)
2030 IF STRIG(0)=0 THEN POKE SNDNUM,MUSTER:REM * Ton ausloesen
2040 POKE CONSOL,8:IF PEEK(CONSOL)=6 THEN GOSUB 7000
2050 IF PEEK(CONSOL)=3 THEN GOSUB 6000
2090 RETURN
3000 REM *** Aendern des momentanen Feldes ***
3020 ON N GOTO 3100,3200,3300,3400,3500,3600,3700,3800,3900,3950
3090 RETURN
3100 REM * Musternummer aendern?
3110 MN=0:MX=31:H=1:A=MUSTER:GOSUB 8000
3120 IF A<>MUSTER THEN MUSTER=A:INDEX=SNDTAB+MUSTER*8:GOSUB 5000
3190 RETURN
3200 REM * Frequenzanfang
3210 R=0:MN=0:MX=255:H=1:GOSUB 8100
3220 RETURN
3300 REM * Frequenzaenderung
3310 R=1:MN=-127:MX=127:H=1:GOSUB 8150:RETURN
3400 REM * Lautstaerkeanfang
3410 R=2:MN=0:MX=248:H=1:GOSUB 8100
3490 RETURN
3500 REM * Lautstaerkeaenderung
```

```
3510 R=3:MN=-127:MX=127:H=1:GOSUB 8150
3590 RETURN
3600 REM * Distortion
3610 R=4:MN=0:MX=192:H=32:GOSUB 8100
3690 RETURN
3700 REM * Laenge des Tones
3710 R=5:MN=0:MX=255:H=1:GOSUB 8100
3790 RETURN
3800 REM * Kanalnummer waehlen
3810 R=6:GOSUB 8200
3820 P=INT(A/4)*4:A=A-P
3830 MN=0:MX=3:H=1:GOSUB 8000
3840 A=P+A:GOSUB 8300
3890 RETURN
3900 REM * Prioritaet einstellen
3910 R=6:GOSUB 8200
3920 P=INT(A/4)*4:K=A-P:A=P/4
3930 MN=0:MX=31:H=1:GOSUB 8000
3940 A=A*4+K:GOSUB 8300:RETURN
3950 REM * Verweis auf naechstes Muster
3960 R=7:GOSUB 8250
3970 MN=-1:MX=31:H=1:GOSUB 8000
3980 GOSUB 8350
3990 RETURN
4000 REM *** Ausgabe des momentanen Feldes auf Bildschirm ***
4020 ON N GOTO 4100,4200,4300,4400,4500,4600,4700,4800,4900,4950
4090 RETURN
4100 REM Musternummer ausgeben
4110 H=MUSTER:L=3:GOSUB 8700
4190 RETURN
4200 REM * Freq. Anfang ausgeben
4210 R=0:L=5:GOSUB 8500:RETURN
4300 REM * Frequenzaenderung ...
4310 R=1:L=6:GOSUB 8600:RETURN
4400 REM * Lautstaerkeanfang
4410 R=2:L=8:GOSUB 8500:RETURN
4500 REM * Lautstaerkeaenderung
4520 R=3:L=9:GOSUB 8600:RETURN
4600 REM * Distortion
4620 R=4:L=11:GOSUB 8200:H=A/32:RESTORE 10001+H:READ D$:POSITION
C,L:? D$;"     ":RETURN
4700 REM * Laenge des Tones
4710 R=5:L=12:GOSUB 8500:RETURN
4790 RETURN
4800 REM * Kanalnummer
4810 R=6:L=13:GOSUB 8200:H=A-INT(A/4)*4:GOSUB 8700:RETURN
4900 REM * Prioritaet
4910 R=6:L=14:GOSUB 8200:H=INT(A/4):GOSUB 8700:RETURN
4950 REM * Verweis auf naechstes Muster
4960 R=7:L=16:GOSUB 8250:IF A=-1 THEN POSITION C,L:? "NIL":RETURN
4970 POSITION C,L:? A;"   ":RETURN
5000 REM *** Gesamter Ausdruck aller Werte eines Musters ***
5010 GOSUB 4100:GOSUB 4200:GOSUB 4300:GOSUB 4400:GOSUB 4500
5020 GOSUB 4600:GOSUB 4700:GOSUB 4800:GOSUB 4900:GOSUB 4950
5090 L=3:RETURN
6000 REM *** SAVE - Routine (abspeichern) ***
6010 D$="SAVE-FNAME (D:FN.EXT)":GOSUB 7500:IF F THEN RETURN
6020 TRAP 6050:OPEN #1,8,0,F$
6030 FOR I=0 TO 255:A=PEEK(SNDTAB+I):PUT #1,A:NEXT I
```

```
6050 CLOSE #1
6060 TRAP 40000:GOSUB 9100:GOSUB 9300
6090 RETURN
7000 REM *** LOAD - Routine (lesen) ***
7010 D$="LOAD-FNAME (D:FN.EXT)":GOSUB 7500:IF F THEN 7050
7020 TRAP 7050:OPEN #1,4,0,F$
7030 FOR I=0 TO 255:GET #1,A:POKE SNDTAB+I,A:NEXT I
7050 CLOSE #1
7060 TRAP 40000:GOSUB 9100:GOSUB 9300
7070 GOSUB 5000
7090 RETURN
7500 REM * Filenamen eingeben
7510 GOSUB 9150:POSITION 2,20:? D$;:POKE 764,255
7520 TRAP 7590:INPUT F$:IF F$(1,1)="D" THEN F=0:RETURN
7590 F=1:RETURN
8000 REM *** Hilfsroutinen, Felder veraendern, Cursor setzen ***
8005 POSITION C-1,L:? CHR$(190);
8010 IF ST=11 THEN A=A-H:IF A<MN THEN A=MN
8020 IF ST=7 THEN A=A+H:IF A>MX THEN A=MX
8050 IF ST<>11 AND ST<>7 THEN GOSUB 9200
8080 POSITION C-1,L:? " ";
8090 RETURN
8100 REM * Wert lesen/aendern/Schreiben (0-255)
8110 GOSUB 8200:GOSUB 8000:GOSUB 8300
8120 RETURN
8150 REM * Wert lesen/aendern/schreiben (+/-127)
8160 GOSUB 8250:GOSUB 8000:GOSUB 8350:RETURN
8200 A=PEEK(INDEX+R):RETURN :REM * Wert lesen (0-255)
8250 A=PEEK(INDEX+R):A=A-256*(A>127):RETURN :REM * lesen +/-127
8300 POKE INDEX+R,A:RETURN :REM * Wert schreiben 0-255
8350 A=A+256*(A<0):POKE INDEX+R,A:RETURN :REM * schreiben +/-127
8500 REM * Wert 0-255 ausdrucken
8510 POSITION C,L:? PEEK(SNDTAB+MUSTER*8+R);"   ":RETURN
8600 REM * Wert +/-127 ausdrucken
8610 GOSUB 8250:POSITION C,L:? A;"   ":RETURN
8700 REM Ausdruck Hilfsvariable H
8710 POSITION C,L:? H;"   ":RETURN
9000 REM * Soundtabelle loeschen
9010 FOR I=0 TO 31
9020 FOR J=0 TO 6:POKE SNDTAB+I*8+J,0:NEXT J:POKE SNDTAB+I*8+7,255
9030 NEXT I
9090 RETURN
9100 REM * Textzeile drucken
9110 POSITION 2,20:? "     >>> TRIGGER FUER SOUND <<<      ":RETURN
9150 REM * Textzeile loeschen
9160 POSITION 2,20:? "                                    ":RETURN
9200 REM * WARTEZEIT
9210 TICK=PEEK(20)
9220 IF PEEK(20)=TICK THEN 9220
9230 RETURN
9300 REM * VBI einschalten
9310 A=USR(SNDEIN)
9320 RETURN
10000 REM *** Bezeichnungen fuer Distortions ***
10001 DATA 5 & 17 Bit Poly
10002 DATA 5 Bit Poly
10003 DATA 5 & 4 Bit Poly
10004 DATA 5 Bit Poly
10005 DATA 17 Bit Poly
```

```
10006 DATA Purer  Ton
10007 DATA 4 Bit Poly
10010 REM ****************************************
32000 REM * SOUND-MASCHINENPROGRAMM
32010 S=0:RESTORE 32100
32020 FOR A=39936 TO 40212:READ D:POKE A,D:S=S+D:NEXT A
32030 IF S<>36579 THEN ? "DATEN-FEHLER!":STOP
32040 SNDEIN=39936:SNDAUS=39940:SNDTAB=40448:SNDFLG=1790:SNDNUM=1791
32090 RETURN
32100 DATA 104,76,222,156,104,76,245,156,216,32,109,156,32,18,156
32110 DATA 76,98,228,169,7,141,244,157,14,254,6,144,6,173,244,157
32120 DATA 32,55,156,206,244,157,16,240,173,255,6,201,255,240,8,32
32130 DATA 55,156,169,255,141,255,6,96,10,10,10,170,189,5,158,240
32140 DATA 44,189,6,158,41,3,168,189,6,158,74,74,217,240,157,144,28
32150 DATA 153,240,157,138,153,228,157,189,0,158,153,232,157,189,2
32160 DATA 158,153,236,157,189,5,158,153,224,157,32,195,156,96,169
32170 DATA 3,141,244,157,32,123,156,206,244,157,16,248,96,172,244
32180 DATA 157,185,224,157,240,63,56,233,1,153,224,157,208,29,190
32190 DATA 228,157,189,7,158,201,255,240,4,32,55,156,96,169,0,153
32200 DATA 240,157,153,232,157,153,236,157,32,195,156,96,190,228,157
32210 DATA 189,1,158,24,121,232,157,153,232,157,189,3,158,24,121,236
32220 DATA 157,153,236,157,32,195,156,96,185,232,157,72,185,236,157
32230 DATA 74,74,74,74,29,4,158,72,152,10,168,104,153,1,210,104,153
32240 DATA 0,210,96,169,0,141,254,6,169,255,141,255,6,32,2,157,162
32250 DATA 156,160,8,169,7,32,92,228,96,162,228,160,98,169,7,32,92
32260 DATA 228,32,2,157,96,169,0,162,7,157,0,210,202,16,250,141,8
32270 DATA 210,169,3,141,15,210,96
```



Verwendung der Sounds in BASIC

Jetzt kommt das Wichtigste: Wie können Sie die mit dem Soundeditor aufbereiteten Töne in Ihr Programm übernehmen? Nun ja, nichts Leichteres als das! Sie brauchen dazu nur den Soundloader 'SNDLOAD.BAS', den Sie gleich im Anschluß abgedruckt finden, an Ihr Programm anhängen. Er erledigt alle Aufgaben, die mit der Einrichtung des Soundprogrammes zusammenhängen: Reservierung des Speicherplatzes, 'Poken' des Maschinenprogrammes, lesen der vom Soundeditor erzeugten Datei und Aktivierung der VBI—Routine.

Der Soundloader

Auf einer der nächsten Seiten sehen Sie ein Listing des BASIC—Loaders für das Sound-Maschinenprogramm SNDLOAD1.BAS. Für diejenigen, die das Programm eintippen wollen ist noch folgendes zu vermerken: Der DATA—Block in SNDLOAD1.BAS ist vollkommen identisch mit dem im Sound—Editor Programm, da schließlich in beiden Programmen dasselbe Maschinenprogramm verwendet wird. Wenn Sie also den Editor schon eingetippt haben, und das ist ja eigentlich Voraussetzung zur Benutzung dieses Programmes, dann können Sie sich das Eingeben wesentlich erleichtern: Laden Sie den Soundeditor, stecken Sie eine leere Diskette in Laufwerk 1 und geben Sie 'LIST "D:S.LST",32000,32299' ein. Jetzt den Sound-Editor mit 'NEW' löschen und die eben auf Disk geschriebenen Programmzeilen mit 'ENTER "D:S.LST"' wieder hereinholen. Na bitte, nun brauchen Sie lediglich noch die restlichen Zeilen einzutippen,

Besitzer der 'Hexenküche'—Diskette sind fein heraus, der BASIC-Loader ist als SNDLOAD1.BAS auf der Diskette enthalten.

Benutzung des Soundloaders

Zuallererst tragen Sie in Zeile 32530 den Filenamen eines mit dem Sound-Editor erzeugten Files ein, z.B. den Namen des im vorigen Kapitel versuchsweise geschriebenen Files TEST.SND. HK-Disk Besitzer geben nichts ein, hier ist bereits der Name des Demo-Files 'DRUMS.SND' eingetragen. Jetzt starten Sie das Programm mit RUN und warten bis die Initialisierung abgeschlossen ist. Sie erhalten jetzt eine Meldung, wie der Sound-VBI wieder abgeschaltet werden kann (A=USR(SNDAUS)) und werden Ihrem Schicksal überlassen.

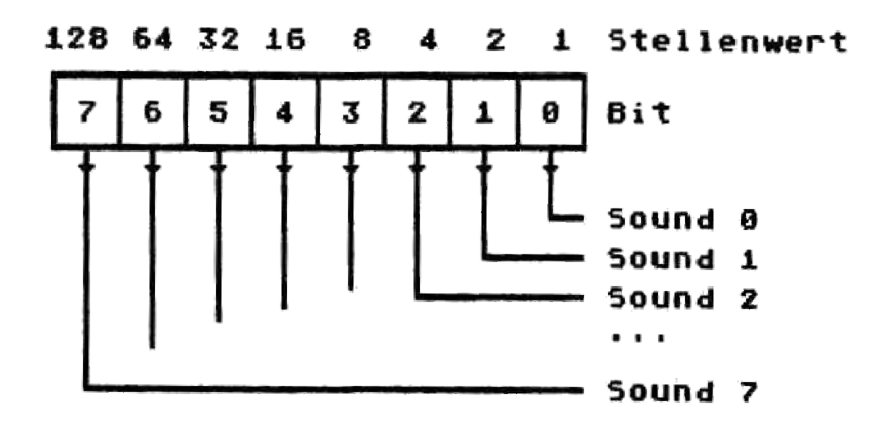
Jetzt kracht's

Sie haben zwei Möglichkeiten einen Sound auszulösen: Die erste und auch einfachere ist, die Nummer des Sounds (0-31) in die Speicherzelle SNDNUM (1790 dez.) zu schreiben. Probieren Sie es gleich aus: Sie tippen

```
POKE SNDNUM,0
```

und schon hören Sie den Sound, der im Editor mit der Musternummer 0 eingestellt wurde. Sie können hier Zahlen von 0-31 benutzen, einen Ton hören Sie dabei natürlich nur dann, wenn auch im Editor einer eingetragen wurde.

Der zweite Weg ist etwas komplizierter, **hat** aber einen wesentlichen Vorteil; Sie **können mehrere** Sounds gleichzeitig auslösen. Ihr ATARI-Computer hat schließlich vier Tonkanäle, und die sollen ja dann auch voll genutzt werden. Die Auslösung erfolgt mittels der Speicherzelle SNDFLG (1791), deren einzelne Bits jeweils einem der Sounds 0-7 entsprechen.



*Soundauslösunq **durch** SNDFLG*

Wollen Sie nur einen Sound, z.B. Sound Nr. 0, hören, so geben Sie

POKE SNDFLG,1

ein. Sind aber mehrere Sounds gleichzeitig gefragt, so müssen Sie deren Stellenwert addieren und die erhaltene Zahl in SNDFLG ablegen: Für Sound 0 (Stellenwert 1) und Sound 2 (Stellenwert 4) bekommen Sie

POKE SNDFLG,5

Das klappt natürlich nur, wenn den Sounds im Sound-Editor such verschiedene Kanäle zugewiesen wurden. Die Möglichkeiten, die Sie mit diesem Auslösemechanismus erhalten sind enorm: Sie können gleichzeitig bis zu vier Soundabfolgen aufrufen. Ein Beispiel für die Anwendung dieser Fähigkeit finden Sie im Abschnitt 'ATARI als Schlagzeuger'. Der Nachteil dieser Auslöseart ist, und das sei nicht verschwiegen, daß Sie damit nur die ersten 8 Sounds benutzen können. Konkret bedeutet das, daß Sie die am häufigst gebrauchten und auch gleichzeitig aufrufbaren Töne beim Editieren in die Positionen 0-7 legen sollten.

```
100 REM * Hier kann Ihr Programm stehen...
110 REM *
120 REM * In Zeile 32530 Sound-Filenamen eingeben!
200 GOSUB 31000:REM * Sound einschalten
210 GRAPHICS 0:? :? "Sie Koennen die mit SNDEDIT erzeugten"
220 ? "nun mit   POKE SNDFLG,<Bitmuster>"
230 ? "   oder   POKE SNDNUM,<Sound-Nummer>"
240 ? "aufrufen. Ausschalten der Sound-"
250 ? "routine mit A=USR(SNDAUS)"
290 END
300 REM *
31000 REM * BASIC-Loader f. Soundroutine
31010 POKE 106,144:GRAPHICS 0:POSITION 10,9:? "Initialisierung..."
31020 GOSUB 32000:REM * Sound-Routine poken
31030 GOSUB 32500:REM * Sound-Tabelle laden
31040 A=USR(SNDEIN):REM * SOUND EINSCHALTEN
31090 RETURN
32000 REM * SOUND-MASCHINENPROGRAMM
32010 S=0:RESTORE 32100
32020 FOR A=39936 TO 40212:READ D:POKE A,D:S=S+D:NEXT A
32030 IF S<>36579 THEN ? "DATEN-FEHLER!":STOP
32040 SNDEIN=39936:SNDAUS=39940:SNDTAB=40448:SNDFLG=1790:SNDNUM=1791
32090 RETURN
32100 DATA 104,76,222,156,104,76,245,156,216,32,109,156,32,18,156
32110 DATA 76,98,228,169,7,141,244,157,14,254,6,144,6,173,244,157
32120 DATA 32,55,156,206,244,157,16,240,173,255,6,201,255,240,8,32
32130 DATA 55,156,169,255,141,255,6,96,10,10,10,170,189,5,158,240
32140 DATA 44,189,6,158,41,3,168,189,6,158,74,74,217,240,157,144,28
32150 DATA 153,240,157,138,153,228,157,189,0,158,153,232,157,189,2
32160 DATA 158,153,236,157,189,5,158,153,224,157,32,195,156,96,169
32170 DATA 3,141,244,157,32,123,156,206,244,157,16,248,96,172,244
32180 DATA 157,185,224,157,240,63,56,233,1,153,224,157,208,29,190
32190 DATA 228,157,189,7,158,201,255,240,4,32,55,156,96,169,0,153
32200 DATA 240,157,153,232,157,153,236,157,32,195,156,96,190,228,157
32210 DATA 189,1,158,24,121,232,157,153,232,157,189,3,158,24,121,236
32220 DATA 157,153,236,157,32,195,156,96,185,232,157,72,185,236,157
32230 DATA 74,74,74,74,29,4,158,72,152,10,168,104,153,1,210,104,153
32240 DATA 0,210,96,169,0,141,254,6,169,255,141,255,6,32,2,157,162
32250 DATA 156,160,8,169,7,32,92,228,96,162,228,160,98,169,7,32,92
32260 DATA 228,32,2,157,96,169,0,162,7,157,0,210,202,16,250,141,8
32270 DATA 210,169,3,141,15,210,96
32400 REM *
32500 REM *** BLOAD ***  Maschinenprogramm initialisieren
32510 BLOAD=1536:RESTORE 32600
32520 FOR A=BLOAD TO BLOAD+33:READ X:POKE A,X:NEXT A
32530 OPEN #3,4,0,"D:DRUMS.SND":REM * HIER SOUND-FILENAMEN EINSETZEN
32540 X=USR(BLOAD,SNDTAB,256):REM *File einlesen
32550 CLOSE #3:RETURN
32600 DATA 104,162,48,169,7,157,66,3,104,157,69,3,104,157,68,3,104
32610 DATA 157,73,3,104,157,72,3,32,86,228,132,212,169,0,133,213,96
```

Das SOUND-Maschinenprogramm

Wichtig für das Verständnis des Sound-Maschinenprogrammes ist die Struktur, mit der die einzelnen Sounds im Speicher abgelegt werden:

1. Byte : Frequenz zum Zeitpunkt des Anstoßes (Freq.Anfang)
2. Byte : Änderung der Frequenz pro 1/50 sec.
3. Byte : Lautstärke zum Zeitpunkt des Anstoßes * 16
4. Byte : Änderung der Lautstärke pro 1/50 sec. * 16
5. Byte : Länge des Sounds in 1/50 sec.
6. Byte : Distortion (Verzerrung)*16
7. Byte : Bit 0-1: Kanalnr. Bit 2-7: Priorität
8. Byte : Verweis auf nächstes Muster

Insgesamt gibt es 32 solcher 8-Byte Blöcke, die ab SNDTAB ($9E00) nacheinander im Speicher liegen, so dass insgesamt 256 Bytes beansprucht werden. Die Soundtabelle an sich ist nicht adressgebunden, sie kann also durchaus auch an eine andere Speicherstelle als im Beispiel des BASIC-Loaders gelegt werden, aber natürlich muss auch das Maschinenprogramm davon Kenntnis nehmen: Die Konstante SNDTAB muss auf den Anfang der Tabelle gerichtet werden. In so einem Fall kann man SNDLOAD1.BAS nicht mehr verwenden, hier muss dann direkt mit dem Assembler-File gearbeitet werden.

Diese Soundtabelle bildet die Datengrundlage des Sound-Maschinenprogrammes, aus ihr werden die benötigten Daten in interne Speicherstellen des VBI-Programmes kopiert und in die momentanen Parameter des Sounds umgerechnet.

Sprungleiste

Das Maschinenprogramm beginnt mit einer sog. Spungleiste aus PLA- und JMP-Befehlen. Das bringt den wesentlichen Vorteil, dass man sich nicht um die internen Programmadressen kümmern braucht, die Aufrufadresse ist mit dieser Methode immer gleich der physikalischen Anfangsadresse des Programmes. Die zweite Einsprungadresse ist damit auch festgelegt, sie ist immer bei der Anfangsadresse + 4. Die PLA-Befehle sind zur Berichtigung des Stacks nach einem Aufruf des Maschinenprogrammes durch BASIC mittels eines USR-Befehles notwendig. Wollen Sie das Soundprogramm aus einem Maschinenprogramm heraus aufrufen, so müssen Sie die PLAs überspringen, Ihr Programm könnte in diesem Fall so aussehen:

```
SNDADR = $9C00 Hier steht Soundprogramm

...

JSR SNDADR+1 ;Sound einschalten

...

JSR SNDADR+5 ;Sound ausschalten

...
```

Initialisierunq
Die Sprungleiste führt beim Einschalten des Sound-Maschinenprogrammes zu dem Unterprogramm SNDINI. Hier wird POKEY zurückgesetzt (in SNDPOK), die Auslöseregister SNDNUM und SNDFLG gelöscht und das Sound-VBI-Programm aktiviert (s. Abschnitt 'Programmierung des VBIs'). Der BASIC-Befehl zum Einschalten des Soundprogrammes lautet: X=USR(39936).

Das Interruptprogramm
beginnt beim Label SNDVBI und gliedert sich in zwei wesentliche Teile: Den Soundinterpreter (SNINTP), der momentan laufende Töne bearbeitet, und den Test auf neue Anforderung (SNDANF), in dem die Übergabespeicherstellen SNDNUM und SNDFLG geprüft werden.

Bleiben wir beim letzteren, Das Unterprogramm SNDANF beginnt mit der Prüfung aller 8 Bits von SNDFLG. Wird eine Eins erkannt, so erfolgt ein UP—Aufruf der Soundvorbereitungsroutine (SNVORB). Anschließend wird nachgesehen, ob in SNDNUM eine Anforderung eingegangen ist (SNDNUM <> 255), wenn ja wird wiederum die Vorbereitungsroutine benutzt.

Im UP SNVORB wird geprüft, ob die Länge des neuen Tones ungleich Null und seine Priorität größer oder gleich der des aktiven Sounds im gewünschten Kanal ist. Trifft beides zu, so werden die Anfangsfrequenz, die Anfangslautstärke und die Tonlänge in die Variablen des jeweiligen Kanals übertragen.

Der Soundinterpreter (SNINTP) besteht zunächst aus einem Rumpfprogramm, welches nur eine Schleife zur Bearbeitung aller vier Kanäle beinhaltet, und aus dem heraus die Interpreterroutine für einen Kanal (SNINTC) viermal aufgerufen wird. Dort wird nachgesehen, ob der momentane Kanal überhaupt aktiv ist. Wenn ja, wird sein Sound-Dauerzähler herunter gezählt und anschließend die Frequenz und Lautstärkeparameter mit Hilfe der Soundtabelle berechnet. Hat sich herausgestellt, dass der Sound zu Ende ist, so wird geprüft, ob eine Soundverkettung vorliegt und eventuell ein weiterer Sound vorbereitet.

Ausschalten
ist ähnlich dem Vorgang des Einschaltens, die Sprungleiste führt zu dem UP SNDEXT, indem der VBI in seinen Urzustand zurückversetzt und POKEY, um laufende Töne zu beenden, nochmals initialisiert wird. Von BASIC wird das mit X=USR(SNDAUS) gesteuert, wobei SNDAUS=39940 ist.

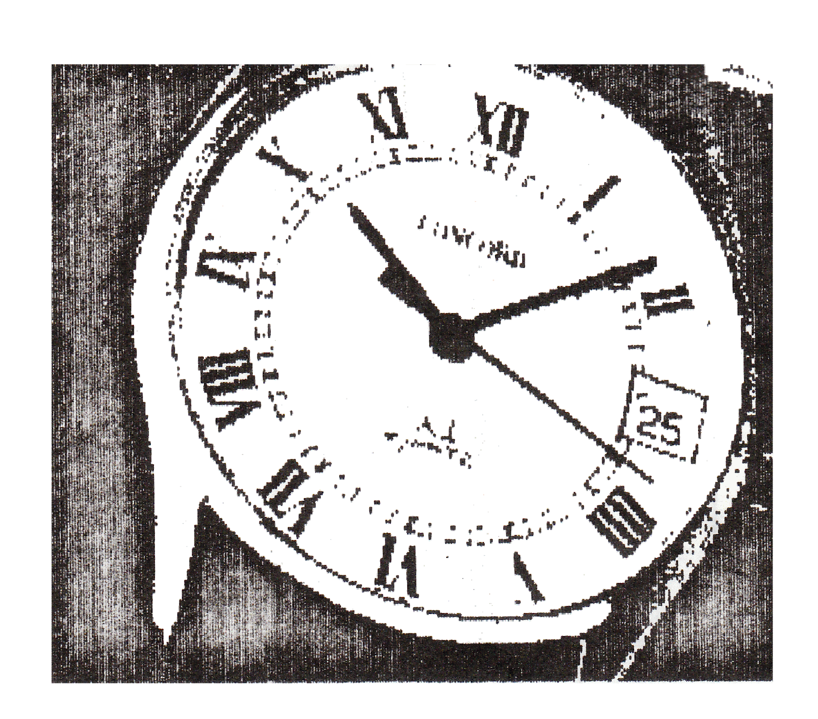
```
0100 ;**************************************************
0110 ;*           SOUND IM VBI                         *
0120 ;*                                                *
0130 ;* Soundgenerator im VBI - zum Editor SNDEDIT    *
0140 ;**************************************************
0150 ;
0160 ; MUSTERTABELLE:
0170 ;
0180 SNDTAB = $9E00  Mustertabelle ist hier abgelegt
0190 ;
0200 ; Struktur der Mustertabelle: wird 32 mal wiederholt
0210 ;
0220 ; 1.Byte: Frequenzanfang
0230 ; 2.Byte: Frequenz Aenderung
0240 ; 3.Byte: Lautstaerke Anfang * 16
0250 ; 4.Byte: Lautstaerke Aenderung *16
0260 ; 5.Byte: Dauer des Sounds in 1/50 sec.
0270 ; 6.Byte: Distortion * 16
0280 ; 7.Byte: Bit 0-1: Kanalnumer, Bit 2-7:Prioritaet
0290 ; 8.Byte: Verweis auf naechstes Muster (255:=NIL)
0300 ;
0310 ;**************************************************
0320 ; OS und Hardwareadressen
0330 ;**************************************************
0340 ;
0350 SETVBV = $E45C  O.S. Vektoren
0360 XITVBV = $E462
0370 ;
0380 AUDF1  = $D200   POKEY-Register
0390 AUDC1  = $D201
0400 AUDCTL = $D208
0410 SKCTL  = $D20F
0420 ;
0430 ;**************************************************
0440 ; Schnittstellen Adressen
0450 ;**************************************************
0460 ;
0470 SNDFLG = $06FE  Aufruf der Sounds 0-7 per Flags
0480 SNDNUM = $06FF  Aufruf der Sounds 0-31 per Nummer
0490 ;
0500 ; Interne Adressen f. SOUND
0510 ;
0520 SNDATA = $9DE0  Anfangsadresse der Variablen
0530 ;
0540 SNDCNT = SNDATA Zaehler f. Soundlaenge (4 Bytes)
0550 SNINDX = SNDATA+4 Index in Mustertabelle (4)
0560 SNFREQ = SNDATA+8 aktuelle Frequenz (4)
0570 SNDVOL = SNDATA+12 aktuelle Lautstaerke (4)
0580 SNDPRI = SNDATA+16 Prioritaet der Sounds (4)
0590 SNDCHN = SNDATA+20 Hilfsregister(1 Byte)
0600 ;
0610 ;
0620 ;
0630 ;**************************************************
0640 ; Sprungleiste zum Aufruf von Basic, Assembler ...
0650 ;**************************************************
0660        *=  $9C00
0670 ;
0680        PLA         fuer BASIC
```

```
0690        JMP SNDINI  SOUND-VBI einschalten, init.
0700        PLA         fuer BASIC
0710        JMP SNDEXT  VBI ausschalten
0720 ;
0730 ;************************************************
0740 ; VBI-Routine, laeuft alle 1/50 sec.
0750 ;************************************************
0760 ;
0770 SNDVBI CLD      ;Dezimalmodus
0780        JSR SNINTP  Interpreter f. laufende Toene
0790        JSR SNDANF  neue Anforderungen pruefen
0800        JMP XITVBV  VBI beenden
0810 ;
0820 ;************************************************
0830 ; Test auf Sound-Anforderung
0840 ;************************************************
0850 ;
0860 SNDANF LDA #7   Anforderung per Flag pruefen
0870        STA SNDCHN  Zaehler f. 8 Bits
0880 ;
0890 SNANF1 ASL SNDFLG Flag in Carry
0900        BCC SNANF2  ist 0, d.h. keine Anforderung -->
0910 ; sonst Sound vorbereiten!
0920        LDA SNDCHN  Nummer des Musters (0-7)
0930        JSR SNVORB  Sound anstossen >>>
0940 ;
0950 SNANF2 DEC SNDCHN schon alle 8 Flags geprueft?
0960        BPL SNANF1  nein, weiter --->
0970 ;
0980        LDA SNDNUM  Anforderung per Musternummer
0990        CMP #255    Code fuer 'keine Anforderung'?
1000        BEQ SNAEND  ja, Schluss -->
1010 ;
1020        JSR SNVORB  sonst Sound vorbereiten >>>
1030        LDA #255    und Flag zuruecksetzen
1040        STA SNDNUM
1050 ;
1060 SNAEND RTS
1070 ;
1080 ;************************************************
1090 ; Soundvorbereitung: Parameter aus Tabelle holen
1100 ; <A>: Musternummer
1110 ;************************************************
1120 ;
1130 SNVORB ASL A    ;Index in Mustertabelle
1140        ASL A       ;= Musternummer * 8
1150        ASL A
1160        TAX         Musterindex in X
1170        LDA SNDTAB+5,X Laenge des Sounds:=0?
1180        BEQ SNVEND  dann kein Ton -->
1190 ;
1200        LDA SNDTAB+6,X Kanal Nr. besorgen
1210        AND #3      ist in Bit 0-1 enthalten
1220        TAY         Kanalnummer nach Y
1230        LDA SNDTAB+6,X zuerst Prioritaet pruefen!
1240        LSR A       Prior. ist Bit 2-7 enthalten
1250        LSR A       d.h. / 4
1260        CMP SNDPRI,Y groesser momentane Prioritaet?
1270        BCC SNVEND  ja, Anforderung wird ignoriert -->
```

```
1280 ;
1290        STA SNDPRI,Y Prioritaet merken
1300        TXA         Index in Mustertabelle
1310        STA SNINDX,Y fuer Interpreter aufbewahren
1320        LDA SNDTAB+0,X Frequenzanfang
1330        STA SNFREQ,Y in Register f. momentane Freq.
1340        LDA SNDTAB+2,X Lautstaerkeanfang
1350        STA SNDVOL,Y Reg. fuer akt. Lautstaerke
1360        LDA SNDTAB+5,X Laenge des Sounds
1370        STA SNDCNT,Y in Zaehler eintragen
1380        JSR SNDHRD  Sound gleich anwerfen >>>
1390 SNVEND RTS
1400 ;
1410 ;*************************************************
1420 ; Soundinterpreter fuer alle 4 Kanaele
1430 ;*************************************************
1440 ;
1450 SNINTP LDA #3   alle 4 Kanaele bearbeiten (0-3)
1460        STA SNDCHN  in Zaehler ablegen
1470 ;
1480 SNNXCH JSR SNINTC einzelnen Kanal bearbeiten >>>
1490        DEC SNDCHN  schon alle 4?
1500        BPL SNNXCH  nein, naechsten Kanal -->
1510        RTS
1520 ;
1530 ;*************************************************
1540 ;Kanal bearbeiten, neue Freq. und Laust. berechnen
1550 ;<SNDCHN>: Kanalnummer
1560 ;*************************************************
1570 ;
1580 SNINTC LDY SNDCHN aktuelle Kanalnummer
1590        LDA SNDCNT,Y Kanalzaehler fuer Soundlaenge
1600        BEQ SNIEND  Kanal ist nicht aktiv -->
1610 ;
1620        SEC         sonst Laenge vermindern
1630        SBC #1      Zaehler dec.
1640        STA SNDCNT,Y ist Zaehler abgelaufen?
1650        BNE SNINOK  nein, dann normale Behandlung ->
1660 ;
1670        LDX SNINDX,Y sonst pruefen, ob Verkettung
1680        LDA SNDTAB+7,X Naechstes Muster
1690        CMP #255    kein weiteres Muster (=NIL)?
1700        BEQ SNDAUS  ja, dann Kanal aus! -->
1710 ;
1720        JSR SNVORB  sonst Sound vorbereiten >>>
1730        RTS
1740 ;
1750 SNDAUS LDA #0   Kanal ausschalten
1760        STA SNDPRI,Y Prioritaet=0
1770        STA SNFREQ,Y Frequenz:= 0
1780        STA SNDVOL,Y Lautstaerke:=0
1790        JSR SNDHRD  an Pokey >>>>
1800        RTS
1810 ;
1820 SNINOK LDX SNINDX,Y Index fuer Muster
1830        LDA SNDTAB+1,X Freq.Aenderung aus Tabelle
1840        CLC
1850        ADC SNFREQ,Y neue Freq. berechnen
1860        STA SNFREQ,Y und abspeichern
```

```
1870        LDA SNDTAB+3,X Laut.Aenderung aus Tabelle
1880        CLC
1890        ADC SNDVOL,Y zu bisheriger Lautst. addieren
1900        STA SNDVOL,Y neues Lautst. speichern
1910        JSR SNDHRD  in Hardware eintragen>>>
1920 SNIEND RTS
1930 ;
1940 ;*****************************************************
1950 ;Lautstaerke und Frequenz in POKEY eintragen
1960 ;<Y>: Kanalnummer    <X>: Musternummer * 8
1970 ;*****************************************************
1980 ;
1990 SNDHRD LDA SNFREQ,Y aktuelle Tonhoehe
2000        PHA         merken
2010        LDA SNDVOL,Y akt. Lautstaerke (*16)
2020        LSR A       dividiert durch 16
2030        LSR A
2040        LSR A
2050        LSR A
2060        ORA SNDTAB+4,X Distortion-Bits dazu
2070        PHA         und auch merken
2080        TYA         Index fuer Hardwareregister
2090        ASL A       ;=<Y> * 2
2100        TAY
2110        PLA         Lautst. und Distortion
2120        STA AUDC1,Y Kanal-Kontrollregister
2130        PLA         Tonhoehe
2140        STA AUDF1,Y Kanal-Frequenzregister
2150        RTS
2160 ;
2170 ;*****************************************************
2180 ; Initialisierungsroutine
2190 ;*****************************************************
2200 ;
2210 SNDINI LDA #0   alle Kanaele ausschalten
2220        STA SNDFLG  Flagaufruf loeschen
2230        LDA #255
2240        STA SNDNUM  Nummeraufruf zuruecksetzen
2250        JSR SNDPOK  POKEY initialisieren
2260 ;
2270        LDX #SNDVBI/256 Hi-Byte der VBI-Routine
2280        LDY #SNDVBI&255 Lo-Byte
2290        LDA #7      fuer 'Deferred VBI'
2300        JSR SETVBV  VBI einfuegen >>>
2310        RTS
2320 ;
2330 ;*****************************************************
2340 ; SOUND-VBI ausschalten
2350 ;*****************************************************
2360 ;
2370 SNDEXT LDX #XITVBV/256 VBI-Vektor restaurieren
2380        LDY #XITVBV&255 Lo-Byte
2390        LDA #7      'Deferred VBI'
2400        JSR SETVBV  und ausschalten >>>
2410        JSR SNDPOK  alle Kanaele ruhig
2420        RTS
2430 ;
```

```
2440 ;*************************************************
2450 ; POKEY ruecksetzen
2460 ;*************************************************
2470 ;
2480 SNDPOK LDA #0   alle Kanaele ausschalten
2490        LDX #7      insgesamt 8 Register
2500 ;
2510 SNDP1 STA AUDF1,X AUDF/C 1/2
2520        DEX         schon alle 4?
2530        BPL SNDP1   nein, weiter -->
2540 ;
2550        STA AUDCTL  Sound-Kontrollregister zurueck
2560        LDA #3      Serial CTL aus
2570        STA SKCTL
2580        RTS
2590 ;
2600        .OPT NO LIST
```



ATARI als Schlagzeuger

Wenn Ihnen der rechte Rhythmus fehlt, dann nehmen Sie doch einfach Ihren ATARI-Computer als Schlagzeuger! Im folgenden Programm sind verschiedene bekannte Rhythmen vorprogrammiert, und mit etwas Geschick können Sie Ihre eigenen Rhythmen oder sogar Schlagzeugsoli selbst eingeben. Wenn Ihnen also bisher ein Rhythmusgerät gefehlt hat. so haben sie jetzt eines: Ihren Computer. Mehr noch: Da ein Computer doch etwas 'intelligenter' als ein einfaches Rhythmusgerät ist, kann er natürlich auch beliebig komplizierte Breaks und Soli spielen!

Was wird gebraucht?

Das Pragramm benutzt die im vorhergehenden Abschnitt erläuterte VEI-Soundroutine, daher

ACHTUNG: zu diesem Programm brauchen Sie unbedingt den weiter vorne beschriebenen Soundeditor SNDEDIT-BAS, da die Klänge der einzelnen Trommeln und Becken damit eingestellt werden..

Im Schlagzeugprogramm wird der Soundloader SNDLOAD1.BAS aus dem letzten Abschnitt verwendet. Wenn Sie den schon eingetippt haben, können Sie ihn gleich als Grundlage verwenden. Die Zeilen 100—250, die ja nur als Demo dienen, müssen Sie allerdings löschen. In Zeile 32530 müssen Sie noch den Filenamen in DRUMS.SND abändern. Das letztere File, welches die Klänge der einzelnen Instrumente enthält, muss mit dem Soundeditor (SNDEDIT.BAS) erzeugt werden. Sie brauchen zu. diesem Zweck nur die folgenden Tabelle Ton für Ton einzustellen:

Muster	0	1	2	3	4	5
Freq. Anfang Freq. Aend.	1 0	18 0	240 0	5 0	2 0	1 0
Lautst. Anf. Lautst. Aend.	96 -13	224 -34	224 -20	138 -13	224 -7	64 8
Modus Länge Kanal Priorität	17Bit 7 0 0	5&17Bit 7 1 0	4Bit 12 2 0	4Bit 11 0 0	17Bit 32 3 0	5&17Bit 17 0 0
Nächst. Must.	NIL	NIL	NIL	NIL	NIL	NIL
Instrument Stellenwert	HiHat 1	Snare 2	Bass 4	Ride 8	Crash 16	HiHat2 32

Wenn Sie alles eingestellt haben, drücken Sie auf die 'OPTION'-Taste und geben den Filenamen D:DRUMS.SND ein.

Auf der HK-Diskstte finden Sie übrigens beide Files bereits fertig enthalten. das Schlagzeugprogramm ist unter dem Namen 'DRUMKIT.BAS' zu finden.

Laden Sie jetzt wieder das Schlagzeugprogramm und starten es mit RUN. Sie können nun unter verschiedenen heißen Rhythmen und einem Solo auswählen und, damit Sie nicht ganz zur Untätigkeit verurteilt sind, können Sie mit einem Joystick in Port 1 die Spielgeschwindigkeit beeinflussen. Drücken Sie den Joystick nach links wird das Tempo verlangsamt, nach rechts wird es schneller.

Selbst programmiertes Schlagzeug

Der Aufruf der einzelnen Instrumente geschieht mit der SNDFLG Variablen, die den gleichzeitigen Anstoß von mehreren Sounds erlaubt. schließlich müssen ja oft mehrere Trommeln oder Becken zum gleichen Zeitpunkt angeschlagen werden. Der Rhythmus wird einfach erzeugt, indem in periodischen Zeitabständen ein neuer 'Anschlagwert' in SNDFLG geschrieben wird. Diese Anschlagwerte sind in DATA-Zeilen (ab Zeile 1000) abgelegt und Sie können diese selbstverständlich frei nach Ihren Wünschen ändern. Den Anschlagwert bekommen Sie durch das Summieren *der* Stellenwerte der einzelnen Instrumente:

Stellenwert: 1 - HiHat (Charleston-Maschine)
2 - Snaredrum (Marschtrommel)
4 - Bassdrum (Baßtrommel)
5 - Ride-Cymbal (Rhythmusbecken)
16 – Crash–Cy'mbal (Betonungsbecken)
32 – HiHat Spezialeffekt

Das Anschlagen von z.B. HiHat mit Bass-Drum gleichzeitig erfordert einen Anschlagwert von

1 + 4 = 5

Weiterhin gibt HiHat mit Snare-Drum einen Anschlagwert von 3, und mit diesen beiden Werten können Sie schon einen einfachen Rhythmus eingeben. Ab Zeile 7010 ist extra Platz für eigene Eingaben reserviert, tippen Sie z.B.

```
7010 DATA 5,0,1,0,3,0,1,0,-1
```

Starten Sie das Programm erneut und wählen nun '7' (Eigenbau) und hören einen relativ einfachen Rhythmus, den Sie natürlich nach Belieben ausbauen können. Die Nullen sind dabei Leerschritte, die für Breaks oder ähnliche Effekte benutzt werden können. Abschluss des Rhythmusblocks bildet die '-1', die veranlasst, dass der Block wieder von vorne begonnen wird.

Und nun viel Spaß beim 'Trommeln'!

```
100 REM ***    ATARI als Schlagzeuger      ***
110 REM ***   benutzt VBI-Sound-Routine    ***
120 REM *** braucht File DRUMS.SND auf D1: ***
130 GOSUB 31000
150 GOSUB 10000:REM * Titel drucken
160 GOSUB 11000:REM * Auswahl
170 GOSUB 200:REM * Spielen
190 GOTO 150
200 REM *** Hier wird getrommelt ***
210 RESTORE RHYT
220 READ A:IF PEEK(764)<>255 THEN RETURN
230 IF A<0 THEN 200
240 POKE SNDFLG,A:FOR T=1 TO DELAY:NEXT T
```

```
250 ST=STICK(0):DELAY=DELAY+(ST=7 AND DELAY<100)-(ST=11 AND DE-
LAY>25)
290 GOTO 220
1000 REM *** Rock 'n' Roll ***
1010 DATA 5,0,1,0,3,0,1,0
1020 DATA 5,0,1,0,3,0,1,2
1030 DATA 5,0,1,0,3,0,1,1
1040 DATA 5,3,5,3,3,3,3,3
1050 DATA 20,0,1,0,3,0,1,0
1060 DATA 5,0,1,0,3,0,1,2
1070 DATA 5,2,5,0,3,0,36,0
1080 DATA 5,0,36,0,7,0,1,1,-1
2000 REM *** Walzer ***
2010 DATA 5,0,3,0,3,0
2020 DATA 5,0,3,0,3,1
2030 DATA 5,0,3,0,3,0
2040 DATA 5,1,3,3,5,3,-1
3000 REM *** Rumba ***
3010 DATA 5,1,1,5,1,1,3,1
3020 DATA 5,1,1,5,1,1,3,1
3030 DATA 5,1,1,5,1,1,3,1
3040 DATA 5,1,1,7,1,1,3,3
3050 DATA 5,1,1,5,1,1,3,1
3060 DATA 5,1,1,5,1,1,3,1
3070 DATA 5,1,1,5,1,1,3,3
3080 DATA 19,3,3,7,1,3,7,3,-1
4000 REM *** Blues ***
4010 DATA 5,0,5,0,1,0,1,0,3,0,1,0
4020 DATA 5,0,5,0,1,1,1,0,3,0,1,5
4030 DATA 5,0,5,0,1,0,1,0,3,0,1,1
4040 DATA 5,1,3,1,1,5,3,3,3,1,3,3,-1
5000 REM *** SLOW-ROCK ***
5010 DATA 12,0,8,0,10,8,12,0
5020 DATA 12,0,8,0,10,0,8,0
5030 DATA 12,8,8,0,10,0,12,8
5040 DATA 20,10,10,10,0,10,10,10,-1
6000 REM *** DRUM-SOLO ***
6010 DATA 2,0,0,0,0,0,2,0,0,0,0,0
6020 DATA 2,0,0,0,2,0,0,0,2,0,0,0
6030 DATA 2,0,0,2,0,0,2,0,0,2,0,0
6040 DATA 2,0,2,0,2,0,2,0,2,0,2,0
6050 DATA 2,2,2,2,2,2,5,2,5,2,5,5
6060 DATA 20,1,1,7,1,1,7,1,1,20,1,1
6070 DATA 7,1,1,7,1,1,20,1,1,7,1,1
6080 DATA 20,2,2,0,2,2,2,0,2,2,2,0
6090 DATA 20,0,2,7,14,8,1,1,14,1,3,3
6100 DATA 14,8,14,8,14,14,20,8,1,38,6,6
6110 DATA 6,6,6,36,6,6,2,2,2,20,2,2
6120 DATA 20,0,0,0,0,0,0,0,0,0,0,0,-1
7000 REM * Eigene Rhythmen
7010 DATA -1
10000 REM *** TITEL DRUCKEN ***
10010 GRAPHICS 2+16:SETCOLOR 2,6,10
10020 ? #6;"  PETER'S DRUM-KIT  ":REM * Kursiv = 'reverse'
10030 POSITION 2,2:? #6;"1 ROCK'N'ROLL"
10040 POSITION 2,3:? #6;"2 BLUES"
10050 POSITION 2,4:? #6;"3 WALZER"
10060 POSITION 2,5:? #6;"4 RUMBA"
10070 POSITION 2,6:? #6;"5 SLOW-ROCK"
```

```
10075 POSITION 2,7:? #6;"6 DRUM-SOLO"
10080 POSITION 2,8:? #6;"7 EIGENBAU"
10085 POSITION 2,9:? #6;"0 SCHLUSS"
10090 POSITION 1,11:? #6;"BITTE NUMMER TIPPEN";
10095 RETURN
11000 REM *** Auswahl ***
11010 OPEN #1,4,0,"K:":GET #1,A:CLOSE #1
11040 IF A=49 THEN RHYT=1010:DELAY=40:RETURN
11050 IF A=50 THEN RHYT=4010:DELAY=50:RETURN
11060 IF A=51 THEN RHYT=2010:DELAY=55:RETURN
11070 IF A=52 THEN RHYT=3010:DELAY=65:RETURN
11080 IF A=53 THEN RHYT=5010:DELAY=60:RETURN
11090 IF A=54 THEN RHYT=6010:DELAY=40:RETURN
11100 IF A=55 THEN RHYT=7010:DELAY=40:RETURN
11110 IF A=48 THEN A=USR(SNDAUS):END
11190 GOTO 11000
31000 REM * BASIC-Loader f. Soundroutine
31010 POKE 106,144:GRAPHICS 0:POSITION 10,9:? "Initialisierung..."
31020 GOSUB 32000:REM * Sound-Routine poken
31030 GOSUB 32500:REM * Sound-Tabelle laden
31040 A=USR(SNDEIN)
31090 RETURN
32000 REM * SOUND-MASCHINENPROGRAMM
32010 S=0:RESTORE 32100
32020 FOR A=39936 TO 40212:READ D:POKE A,D:S=S+D:NEXT A
32030 IF S<>36579 THEN ? "DATEN-FEHLER!":STOP
32040 SNDEIN=39936:SNDAUS=39940:SNDTAB=40448:SNDFLG=1790:SNDNUM=1791
32090 RETURN
32100 DATA 104,76,222,156,104,76,245,156,216,32,109,156,32,18,156
32110 DATA 76,98,228,169,7,141,244,157,14,254,6,144,6,173,244,157
32120 DATA 32,55,156,206,244,157,16,240,173,255,6,201,255,240,8,32
32130 DATA 55,156,169,255,141,255,6,96,10,10,10,170,189,5,158,240
32140 DATA 44,189,6,158,41,3,168,189,6,158,74,74,217,240,157,144,28
32150 DATA 153,240,157,138,153,228,157,189,0,158,153,232,157,189,2
32160 DATA 158,153,236,157,189,5,158,153,224,157,32,195,156,96,169
32170 DATA 3,141,244,157,32,123,156,206,244,157,16,248,96,172,244
32180 DATA 157,185,224,157,240,63,56,233,1,153,224,157,208,29,190
32190 DATA 228,157,189,7,158,201,255,240,4,32,55,156,96,169,0,153
32200 DATA 240,157,153,232,157,153,236,157,32,195,156,96,190,228,157
32210 DATA 189,1,158,24,121,232,157,153,232,157,189,3,158,24,121,236
32220 DATA 157,153,236,157,32,195,156,96,185,232,157,72,185,236,157
32230 DATA 74,74,74,74,29,4,158,72,152,10,168,104,153,1,210,104,153
32240 DATA 0,210,96,169,0,141,254,6,169,255,141,255,6,32,2,157,162
32250 DATA 156,160,8,169,7,32,92,228,96,162,228,160,98,169,7,32,92
32260 DATA 228,32,2,157,96,169,0,162,7,157,0,210,202,16,250,141,8
32270 DATA 210,169,3,141,15,210,96
32500 REM *** BLOAD ***  Maschinenprogramm initialisieren
32510 BLOAD=1536:RESTORE 32600
32520 FOR A=BLOAD TO BLOAD+33:READ X:POKE A,X:NEXT A
32530 OPEN #3,4,0,"D:DRUMS.SND":REM * Hier Sound-Filenamen einsetzen
32540 X=USR(BLOAD,SNDTAB,256):REM *File einlesen
32550 CLOSE #3:RETURN
32600 DATA 104,162,48,169,7,157,66,3,104,157,69,3,104,157,68,3,104
32610 DATA 157,73,3,104,157,72,3,32,86,228,132,212,169,0,133,213,9
```

Wie schnell ist Ihr ATARI?

"Mit 3,0 MHz Prozessor" - "1.79 MHz Taktfrequenz" - "30% schneller wenn Bildschirm ausgeschaltet" - Das sind alles Aussagen, die Sie bestimmt schon in einem Prospekt oder in Zeitschriften gelesen haben. Aber - wie schnell läuft Ihr Atari denn nun wirklich? Kann er schnel1er rechnen als z.B. ein C-64 oder ein Apple?

Beginnen wir mit dem Einfachsten: Das Herz Ihres Atari-Computers ist ein 6502 Mikroprozessor, der gleiche Chip der im Apple und auch im C—64 seinen Dienst versieht. Im C-64 ist es zwar eigentlich ein 6510, der aber bis auf ein paar Kleinigkeiten mit einem 6502 identisch ist. Die Arbeitsgeschwindigkeit des Prozessors hängt von seiner Taktfrequenz ab, die man sich anschaulich als Grundrhythmus seiner Arbeitsweise vorstellen kann. Pro Taktperiode ('Cycle', Zyklus) kann ein Speicherzugriff oder auch eine prozessorinterne Operation stattfinden. Ihr Atari läuft mit einer Taktfrequenz von 1,79 MHz, kann also fast zwei Millionen Zyklen pro Sekunde hinter sich bringen. Wenn man bedenkt, daß der einfachste Additionsbefehl zwei derartiger Zyklen benötigt, so könnte (!) Ihr Atari immerhin fast eine Million Additionen in der Sekunde bewältigen. wir können jetzt schon einen ersten Vergleich anstellen?

Apple	: 1.023 MHz
ATARI	: 1.79 MHz
Commodore 64	: 0.98 MHz

Danach gemessen wäre der Atari-Computer beinahe doppelt so schnell wie ein Apple oder ein C—64. Leider, wie so oft im Leben, trügt der erste Schein. Es gibt nämlich zwei Faktoren, die den Prozessor davon abhalten, so schnell zu rechnen wie er eigentlich könnte: den Refresh und die DMA.

Speicher mit Kurzeitqedächtnis - Refresh

Der Speicher in Ihrem Atari ist aus sog. 'dynamischen RAMs' aufgebaut. Das sind Speicherbausteine, die ihre Informationen eigentlich nur für kurze Zeit (einige tausendstel Sekunden) behalten können. Für die elektronisch Angehauchten unter Ihnen sei gesagt, daß es sich dabei um Kondensatorladungen handelt, die durch nicht zu vermeidende Widerstände ständig verkleinert werden. Um diesen Effekt als vernünftigen Datenspeicher zu verwenden, muss man zu einem Trick greifen: Der ganze Speicher wird periodisch 'aufgefrischt' (daher 'Refresh'), indem er blockweise gelesen wird. Der Refresh-Vorgang kostet natürlich Zeit, die von der Rechenzeit des 6502 mittels sogenannten 'Cycle Stealing' abgezweigt wird. Konkret heißt das, wenn ein Refresh-Zyklus notwendig ist, wird *der* 6502 kurzerhand für einen Zyklus angehalten und ihm auf diese Weise der Zyklus gestohlen. Die Kontrolle des Refreshs übernimmt der DMA-Baustein ANTIC, daher ist es auch nicht weiter verwunderlich, dass der Refresh eng mit dem Bildschirmaufbau verknüpft ist: Im Normalfall werden pro Bildschirmrasterzeile neun Refreshzyklen ausgeführt. Normalfall deswegen, da es bei bestimmten Graphikmodi Ausnahmen gibt.

Ein komplettes Bild besteht aus 312 Rasterzeilen (nicht 192, so viele sind am Fernsehgerät darstellbar!); je 50 Bilder werden pro Sekunde generiert, also sind

9 x 312 x 50 = 140400 Zyklen

für den Refresh nötig. Tatsächlich sind es noch ein paar mehr, da der Refresh auch während der Vertical—Blank Phase stattfindet, was aber gegenüber der obigen Zahl getrost vernachlässigt werden kann. Es bleiben noch

1.790.000 - 140.400 = 1.649.600 Zyklen

zum Rechnen übrig, was einer theoretischen 'Taktfrequenz' von ca. 1.65 MHz entspricht.

Graphik kostet Zeit - die DMA

Soll etwas am Bildschirm angezeigt werden - und das ist ja wohl meistens der Fall — dann müssen Bildinformationen aus dem Speicher in das Videosignal gelangen. Dieser Vorgang wird per DMA (Direct Memory Access) erledigt, darunter versteht man einen Vorgang, der Daten in oder aus dem Speicher transferiert, ohne dass der Prozessor damit aktiv beschäftigt ist. Wohlgemerkt, der Prozessor ist zwar nicht mit dem Datentransfer beschäftigt, wird aber durch ihn beeinflusst. Das Prinzip ist das schon beim Refresh besprochene 'Cycle Stealing', so dass pro mit DMA gelesenen Byte dem 6502 ein Zyklus entwendet wird. Die Berechnung der auf die Bildschirm-DMA entfallenden Zyklen ist allerdings nicht mehr so einfach, da die Anzahl der zu lesenden Bytes stark vom Graphikmodus abhängt. Und Graphikmodi gibt es ja beim Atari mehr als genug. Den mitunter schlimmsten DMA-Bedarf hat der am häufigsten gebrauchte Textmodus GRAPHICS 0. Hier werden ganze 433.400 Zyklen pro Sekunde (s. Kasten) für den Bildschirmaufbau benötigt. Bilden wir jetzt die Bilanz, so bleiben uns noch

1.790.000 - 140.400 - 433.400 = 1.216.200 Zyklen.

Zyklen fuer			
Displaylist	:	1	
Chr. Nummern lesen	:	40	
Zeichensatz lesen	:	8x40	
24 Zeilen/Bild		361	X 24
	=	8664	
Rest Disp.-List	+	8	
50 Bilder/sec.	=	8672	x 50
Zyklen/sec: =		**433.400**	

Berechnung der auf DMA entfallenden Zyklen in GRAPHICS 0

So ganz stimmt unsere Rechnung allerdings immer noch nicht. da, wie oben schon bemerkt, der Refresh nicht immer mit seinen 9 Zyklen zum Zuge kommt. Genau das trifft auch bei GRAPHICS 0 zu, so dass in der obersten Rasterzeile jeder Textzeile nur ein einziger Refreshzyklus ausgeführt wird. Der Grund für dieses Verhalten ist darin zu suchen, dass in dieser Zeile das größte DMA- 'Gedränge' herrscht, *da* dort sowohl die Nummern der Zeichen als auch die erste Zeile des Zeichensatzes gelesen werden müssen. Von den ursprünglich 114 Zyklen, die pro Rasterzeile zur Verfügung stehen, bleiben in einer derartigen Zeile gerade noch (im schlimmsten Fall) 30! Insgesamt werden (8x24 Zeilen x50 Bilder=) 9600 Refreshzyklen pro Sekunde 'eingespart', so dass sich die Zahl der auf den Refresh entfallenden Zyklen auf 130.800 vermindert.

Es geht noch weiter

Eine weitere Ursache für die Verlangsamung der Rechengeschwingigkeit ist in der Systemsoftware zu finden: Der VBI (s. Abschnitt über VBI). Er wird 50 mal in der Sekunde angestoßen und benötigt jeweils ca. 1000 Zyklen (natürlich nur, wenn Sie keine Zusatzroutinen in den VBI eingefügt haben). Summa summarum ergibt das weitere 50.000 Zyklen, in denen uns der Prozessor nicht zur Verfügung steht. Endgültiges Fazit: Wir haben pro Sekunde

1,790.000 - 130.800 - 433.400 - 50.000 = 1.175.800 Zyklen
(Zyk./Sec) (Refresh) (DMA) (VBI)

zur Verfügung, das entspräche einem 6502—Prozessor, der mit ca. 1.17 MHz getaktet wird. Streng genommen kann man natürlich nicht mehr von einer Taktfrequenz sprechen, da die Verzögerungsereignisse (vom Prozessor aus gesehen) in relativ unregelmäßigen Abständen erfolgen. Nebenbei bemerkt, ist das auch der Grund, warum der Sprachsynthesizer 'S.A.M.' von Don't Ask Software bei eingeschalteter DMA und VBI recht heiser klingt.

Falls Sie die Player-Missile DMA eingeschaltet haben, schlägt das übrigens nochmal mit 64000 Zyklen pro Sekunde zu Buche.

Zyklen des Testprogrammes: 59,136.900

rechnerische Taktfreq./MHz (Laufzeit)

	ATARI	C-64	APPLE
Textmodus VBI aktiv	**1.16** (50..9sec)	**0.92** (64.6sec)	**1.023** (60.5sec)
DMA aus MBI aus	**1.62** (36.4sec)	**0.98** (60.3sec)	./.

Vergleich der Laufzeiten

Um dieser grauen Theorie der letzten Seiten etwas Farbe zu geben, wurden Lautzeitmessungen an verschiedenen 6502 Computern vorgenommen, deren Ergebnisse in der obigen Tabelle vergleichend gegenübergestellt wurden. Zum Test wurde ein kurzes Maschinenprogramm verwendet, das am Ende dieses Kapitels abgedruckt ist. Angehende Maschinenprogrammierer können daran gleich sehen, wie bei einem Assemblerprogramm die Gesamtanzahl der Zyklen berechnet wird.

Sie sehen, Ihr Atari-Computer ist damit immer noch um ca. 26% schneller als ein Commodore 64, schaltet man jedoch bei beiden Computern Bildschirm und VBI aus, so liegt der Atari sogar mit 65% vorne.

übrigens - um Missverständnissen vorzubeugen - hat die Prozessorgeschwindigkeit nur mittelbar mit der Rechengeschwindigkeit in BASIC zu tun. Hier spielen noch andere Faktoren wie z.B. Rechengenauigkeit und Art der Arithmetik (BCD bzw. Binär) eine ausschlaggebende Rolle.

Hier noch das Laufzeit—Testproqramm:

```
            0100 ;*******************************************
            0110 ;Testprogramm zur Laufzeitermittlung
            0120 ;*******************************************
            0130 ;
     =00CD  0140 COUNT  =   $CD     Zeropage-Hi1fsregister
            0150 ;
0000        0160        *=  $0600
            0170 ;
0600 A964   0180 START  LDA #100    ;*2* 100 Durchlaeufe
0602 S5CD   0190        STA COUNT   ;*3*
0604 A000   0200 SCHLF1 LDY #0      ;*2* 256 Durchlaeute++++++++
0606 A200   0210 SCHLF2 LDX #0      ;*2* ==================== +
0608 EA     0220 SCHLF3 NOP         ;*2* ------------------ = +
0609 EA     0230        NOP         ;*2* innere Schleife   - = +
060A CA     0240        DEX         ;*2* 9 Zyklen          - = +
060B DOFB   0250        BNE SCHLF3  ;*2/3* -----------------= +
            0260 ;
060D 88     0270        DEY         ;*2* mittlere Schleife   = +
060E D0F6   0280        BNE SCHLF2  ;*2/3* ================== +
            0290 ;
0610 C6CD   0300        DEC COUNT   ;*5* aeussere Schleife   = +
0612 DOFO   0310        BNE SCHLF1  ;*2/3* +++++++++++++++++++++
            0320 ;
0614 60     0330        RTS         ;*6*
            0340 ;
            0350 ;ANZAHL DER ZYKLEN:
            0360 ;-------------------
            0370 ;
            0380 ;Innere Schleife: (9*256)-1=2303
            0385 ;  (BNE nur 2 Zykl. beim letzten Durchlauf)
            0390 ;mittlere Schleife: (2303+7)#256-1=591.359
            0400 ;aeussere Schleife: (591359+10)#100-1=59.136.899
            0410 ;Sonstiges        : 59136699+11=59.136.910
            0420 ;
            0430 ;Summa Summarum: ca.. 59.136.900 Zyklen
            0440 ;
```

Drei neue Grafikmodi

Sie haben richtig gelesen - außer den 17 Graphikmodi die Ihr ATARI bisher schon beherrschte - schlummern noch weitere drei in den elektronischen Tiefen Ihres Computers. Damit haben Sie ganze 20 Graphikmodi zur Verfügung die zu allem Überfluss fast beliebig miteinander mischbar sind. Welcher andere Homecomputer kann da noch mithalten?

Die drei neuen Graphikstufen sind sogenannte Charaktermodi, d.h. sie sind aus einzelnen Zeichen aufgebaut und benötigen einen speziellen Zeichensatz. Das Besondere an diesen Graphikstufen ist, dass ein Zeichen bis zu 16 unterschiedliche Farben gleichzeitig enthalten kann, und ein ganzer Bildschirm davon nur weniger als ein KByte Speicher in Anspruch nimmt.

Um den neuen Kindern gleich einen Namen zu geben, werden sie im folgenden als Graphics 9/0, 10/0 und 11/0 bezeichnet. Warum gerade diese Bezeichnungen gewählt wurden, werden Sie gleich im Anschluss sehen. Die Daten der Graphikstufen im Einzelnen:

Modus	Zeichen pro Zeile	Höhe in Rasterzeilen	Verfügbare Farben
GR. 9/0	40	8	16 Hellikeits-stufen einer Farbe
GR.10/0	40	8	9 Farben durch Farbregister
GR.11/0	40	8	16 Farben einer Helligkeit

Jetzt haben Sie die Lunte vielleicht schon gerochen und erkannt, dass dabei der 'GTIA' seine Finger im Spiel haben muss - richtig geraten, aber zuerst ein

Blick ins Innenleben

Die Vielzahl der möglichen Bildschirmgraphiken des Atari-Computers wird durch die enge Zusammenarbeit zweier Chips, ANTIC und GTIA, ermöglicht. ANTIC, fast schon eine Art 'Video—Mikroprozessor', besorgt die Bildschirmdaten per DMA, und gibt diese an GTIA weiter, der seinerseits die Farben beisteuert und letztendlich auch das Videosignal erzeugt. Aus dieser Anordnung ergeben sich zwei grundsätzlich unterschiedliche Arten von Graphikmodi: Zum einen die sogenannten ANTIC-Modi, 14 an der Zahl, diese werden durch die Displaylist bestimmt, zum anderen die drei GTIA-Modi, die entstehen, indem ein und derselbe ANTIC-Modus vom GTIA unterschiedlich interpretiert wird.

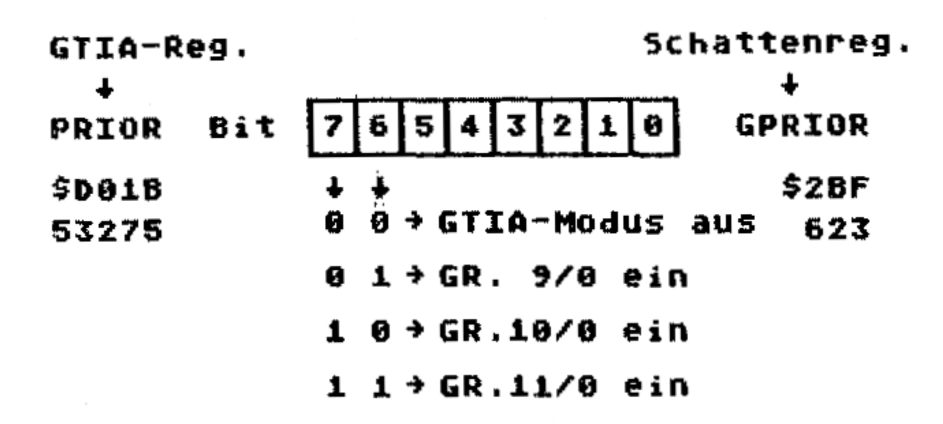
Sehen wir uns zuerst die BASIC-Graphikstufen GRAPHICS 9, 10 und 11 an, denn dabei handelt es sich um die drei eben erwähnten GTIA-Modi. Die Auflösung beträgt in allen drei Fällen 80(H) x 192(V) , der Speicherplatzbedarf liegt bei 7680 Bytes (plus Displaylist), genauso viel wie für GRAPHICS 8, der hochauflösenden Graphik mit 320(H) x 192(V) Punkten benötigt werden.

Das ist auch nicht weiter verwunderlich, denn in GR. 9/10/11 wird derselbe ANTIC-Modus (ANTIC "F") wie auch für GR. 8 verwendet, nur werden die von ANTIC gelieferten Daten eben anders interpretiert. Bei GR. 8 wird jedes einzelne Bit als Heilligkeitsinformation eines Pixels gewertet, GR. 9 hingegen fasst jeweils vier aufeinander folgende Bits zu einer detaillierteren Helligkeitsinformation für ein ausgedehnteres Pixel zusammen, so dass 16 Helligkeitsstufen einer Farbe zur Verfügung stehen. Dadurch schränkt sich natürlich die horizontale Auflösung von 320 auf 80 Pixels pro Zeile ein, die vertikale Auflösung bleibt jedoch nach wie vor bei 192 Pixels. Dieser Modus, eigentlich nicht mehr als hochauflösend zu bezeichnen, eignet sich aber hervorragend für die Darstellung digitalisierter Bilder, der Eindruck ist durch die Möglichkeit der Schattierung ungeheuer plastisch. Mit der gleichen Methode werden die Modi 10 und 11 erzeugt, nur wird hier die 4-Bit Information im einen Fall als Auswahl unter 16 Farben einer Heiligkeit (GR.11), im anderen Fall als Auswahl zwischen 9 Farbregistern (GR.10) interpretiert. Beim letzteren Modus wären theoretisch 16 Farben möglich, da aber nur 9 Farbregister in der Hardware vorhanden sind, muss man sich eben mit diesen begnügen.

Der springende Punkt
der obigen Überlegung ist, dass diese drei Modi durch einfaches Umschalten des GTIA-Chips aus ein und derselben ANTIC-Graphikstufe "F" erzeugt werden. Die gleiche Pixelauflösung, nur eben als Charactermodus, erzeugt der ganz normale Textbildschirm GRAPHICS 0 (ANTIC-Mode 2). Der einzige Unterschied zwischen GR.8 und GR.0 liegt in der Mehrarbeit, die ANTIC bei letzterer aufbringen muss, da die Pixelinformation anstatt direkt aus dem Bildschirmspeicher nun über den Umweg eines Zeichensatzes kommt. Es liegt nun nahe zu fragen, ob man nicht auch bei GR.0 die drei GTIA-Modi verwenden kann — und — es funktioniert! Wenn Sie es gleich ausprobieren wollen, brauchen Sie nur (in BASIC) den Befehl

```
POKE 623,128
```

eingeben und haben damit GRAPHICS 10/0 eingeschaltet! Die Speicherzelle 623 ist der 'Schatten' des GTIA-Registers PRIOR, dessen Bits 6-7 zur Umschaltung der GTIA-Modi benutzt werden:



Hardware-/ Schattenreqister für GTIA-Modi

Sie können die neuen Graphikmodi in BASIC mit folgenden Befehlen erreichen:

```
POKE 623, 64 -> GRAPHICS 9/0
POKE 623,128 -> GRAPHICS 10/0
POKE 623,192 -> GRAPHICS 11/0
```

Versuchen Sie, anstatt des Schattenregisters gleich das Hardwareregister PRIOR selbst umzuschalten, so werden Sie nur mit einem kurzen Aufblitzen der GTIA-Graphik belohnt: Innerhalb von l/50sec. wird PRIOR durch seinen alten Wert im Schattenregister überschrieben. Änderung von PRIOR direkt kann aber auch seine guten Seiten haben: Benutzt man einen Displaylist-Interrupt zur Änderung von PRIOR, so kann man GTIA-Modi mit normalen ANTIC-Modi mischen...

Wenn Sie jetzt einige Buchstaben eintippen, so sehen Sie buntschillernde abstrakte Zeichen, die mit ihren früheren GR.0 Aussehen nichts mehr gemeinsam haben, und hier liegt das Problem: Um diese Graphikmodi vernünftig zu nützen, braucht man einen speziell zugeschnittenen Zeichensatz. Ein Zeichen besteht jetzt aus 2(H) x 8(V) Matrix und sieht so aus:

Bit:	7 6 5 4	3 2 1 0	-> je 4 Bit
			bestimmen:
			- GR. 9/0:
			Helligkeit
			- GR.10/0:
			Farbregister
			- GR.11/0:
			Farbe

Neue Zeichenmatrix

Damit lassen sich zwar keine Buchstaben und Zahlen mehr darstellen, aber für vielfarbige Graphikzeichen ist diese Art von Character-Modus doch sehr nützlich: Bei geschickter Wahl des Zeichensatzes können Sie Bilder, die vorher in den Bit-Map Graphikstufen 9/10/11 fast 8 KByte beansprucht haben, in ein KByte packen. Oder stellen Sie sich scrollende Spielfelder, die viele Bildschirminhalte groß sein können, in 16 leuchtenden Farben vor...

Beispiele — Beispiele — Beispiele

In Listing 1 ist ein Beispielprogramm für GR.11/0 mit 15 Farbsäulen (plus Hintergrundfarbe) gegeben. Dazu wird (ab Zeile 1000) ein Zeichensatz erzeugt, der den Buchstaben A-O jeweils einen Block mit einer Farbe zugeordnet. Wollen Sie selbst etwas experimentieren, so brauchen Sie das Programm nur mit "BREAK" anzuhalten, und können die neuen Zeichen mit den Tasten A-0 aufrufen.

Die Helligkeit der Darstellung können Sie mittels

SETCOLOR 4,0,<Helligkeit> ;Helligkeit von 0-14

einstellen (im Beispiel: Zeile 90), wichtig ist nur, dass Sie als Farbe eine Null eintragen, da die tatsächliche Farbinformation eines Pixels durch logische ODER–Verknüpfung aus dem hier eingetragen Wert und den vier Pixel–Bits ermittelt wird.

Zurück in den normalen Textmodus kommen Sie am einfachsten mit 'System–Reset'.

```
10 REM **** GRAPHICS 11/0 DEMO ****
20 POKE 106,144:GRAPHICS 0:REM * Platz fuer Zeichensatz reservieren
30 CHRSET=(9*16+12)*256:REM * Adresse des neuen Zeichensatzes
40 POKE 752,1:? "}":REM * Cursor aus; '}':= Esc CTRL-Clear
50 GOSUB 1000:REM Zeichensatz aufbauen
60 POKE 756,CHRSET/256:REM * Auf neuen Zeichensatz umschalten
80 POKE 623,192:REM >>>> Graphics 11/0 aufrufen <<<
90 SETCOLOR 4,0,8:REM * Helligkeit einstellen
100 FOR I=0 TO 15:REM * Bildschirmmuster zeichnen
110 COLOR ASC("A")+I:X=I*2+5:PLOT X,0:DRAWTO X,23
120 NEXT I
190 GOTO 190:REM * um Graphik zu erhalten
1000 REM * GR.11/0 Zeichensatz aufbauen
1010 FOR Z=0 TO 15:REM * 16 neue Zeichen
1020 OFFSET=(Z+32)*8:REM * Position im Speicher
1030 FMUST=Z*17:REM * Farbmuster
1040 FOR Z1=0 TO 7:POKE CHRSET+OFFSET+Z1,FMUST:NEXT Z1
1050 NEXT Z
1090 RETURN
```

Listing 1: Demo für GRAPHICS 11/0

Im Modus GR.10/0 kann man schon recht effektive Farbregisteranimation betreiben, ein Beispiel finden Sie in Listing 2. Das Prinzip ist recht einfach: Die Bitmuster im Zeichensatz beschreiben nicht wie bei GR.9/0 eine feste Farbe. sondern verweisen indirekt auf den Inhalt eines Farbregisters. Zeichnen Sie nun ein geeignetes Bild und lassen einen Wert durch die Farbregister laufen, so sieht man nur diejenigen Bitmuster, deren zugehöriges Farbregister gerade die Farbinformation enthält. Ähnlich wird im Programm von Listing 2 verfahren, nur dass hier ein ganzer Strom von Farbwerten durch die Farbregister wandert. dadurch wird ein psychedelischer Effekt erreicht, der an einen Flug durch ein Tunnel erinnert.

Sie können diese neuen Graphikmodi auch mit anderen mischen, obwohl dies nicht so einfach ist wie bei den normalen ANTIC-Modi. Das können Sie schon aus der Tatsache ersehen, dass es in den regulären Graphikstufen GR.9-11 kein Textfenster, wie z.B. bei GR.8 gibt (lesen Sie dazu auch den nächsten Abschnitt!).

Das Demo-Programm in Listing 3 zeigt Ihnen, wie Sie in GR.9/0 ein unabhängiges, vierzeiliges Textfenster einrichten können.

```
10 REM *** GRAPHICS 10/0 DEMO - Farbregisteranimation ***
20 POKE 106,144:GRAPHICS 0:REM * Platz fuer Zeichensatz reservieren
30 CHRSET=(9*16+12)*256:REM * Adresse des neuen Zeichensatzes
40 POKE 752,1:REM * Cursor aus
50 GOSUB 1000:REM * Zeichensatz aufbauen
60 POKE 756,CHRSET/256:REM * Auf neuen Zeichensatz umschalten
80 POKE 623,128:REM >>>> Graphics 10/0 aufrufen <<<<
90 FOR I=0 TO 8:POKE 704+I,I*2:NEXT I:REM * Bildschirm aufbauen
100 FOR I=0 TO 11:COLOR ASC("A")+I-8*(I>7)
110 PLOT I,I:DRAWTO 39-I,I:DRAWTO 39-I,23-I:DRAWTO I,23-I:DRAWTO I,I
120 NEXT I
150 FOR I=0 TO 8:A=712-I:C=PEEK(A)+2:IF C>255 THEN C=0
160 POKE A,C:NEXT I:REM * Farbregister 'umschaufeln'
180 FOR T=0 TO 10:NEXT T
190 GOTO 150
1000 REM * GR.10/0 Zeichensatz aufbauen
1010 FOR Z=0 TO 15:REM * 16 neue Zeichen
1020 OFFSET=(Z+32)*8:REM * Position im Speicher
1030 FMUST=Z*17:REM * Farbmuster
1040 FOR Z1=0 TO 7:POKE CHRSET+OFFSET+Z1,FMUST:NEXT Z1
1050 NEXT Z
1090 RETURN
```

Listing 2: Farbregisteranimation in GR 10/0

```
10 REM *** GRAPHICS 9/0 DEMO mit TEXTFENSTER ***
20 POKE 106,144:GRAPHICS 0:REM * Platz fuer Zeichensatz reservieren
30 CHRSET=(9*16+12)*256:REM * Adresse des neuen Zeichensatzes
40 POKE 752,1:REM * Cursor aus
50 GOSUB 1000:REM * Zeichensatz aufbauen
60 GOSUB 3000:REM DLI-Maschinenprogramm 'poken'
70 GOSUB 2000:REM DLI aktivieren
80 POKE 756,CHRSET/256:REM * Auf neuen Zeichensatz umschalten
90 POKE 623,64:REM Graphics 9/0 aufrufen
100 SETCOLOR 4,2,0:REM * Farbe einstellen
110 FOR I=0 TO 19:REM * Bildschirmmuster zeichnen
120 POSITION 0,I:? "KLMNONMLKJIHGFEDCBA ABCDEFGHIJKLMNONMLKJ";
130 NEXT I
140 POKE 703,4:REM * Textwindow einschalten
150 POKE 752,0:REM Cursor wieder ein
160 LIST :END
1000 REM * GR.9/0 Zeichensatz aufbauen
1010 FOR Z=0 TO 15:REM * 16 neue Zeichen
1020 OFFSET=(Z+32)*8:REM * Position im Speicher
1030 FMUST=Z*17:REM * Farbmuster
1040 FOR Z1=0 TO 7:POKE CHRSET+OFFSET+Z1,FMUST:NEXT Z1
1050 NEXT Z
1090 RETURN
2000 REM * DLI fuer Textfenster aktivieren
2010 DLIVEC=512:NMIEN=54286:SDLDTL=560
2020 DLIST=PEEK(560)+PEEK(561)*256:REM Displaylistadresse
2030 POKE DLIST+24,2+128:REM Bit fuer DLI setzen
2040 POKE DLIVEC,0:POKE DLIVEC+1,6:REM * DLI-Vektor richten
2050 POKE NMIEN,192:REM DLI wird aktiv
2090 RETURN
3000 REM * DLI-Maschinenprogramm einrichten
3010 RESTORE 3100
3020 FOR A=1536 TO 1558:READ D:POKE A,D:NEXT A
3090 RETURN
3100 DATA 72,138,72,173,111,2,41,63,162,224,141,10,212,141,27,208
3110 DATA 142,9,212,104,170,104,64
```

Listing 3: GR. 9/0 mit Textfenster

Die Farbe der Darstellung wird in Zeile 100 mit SETCOLOR4,2,0 eingestellt. Wichtig ist hier wiederum, dass Sie als Helligkeit eine Null eintragen, da sich die Helligkeit eines Pixels durch bitweise ODER-Verknüpfung dieses Wertes mit seinen vier Pixel-Bits ergibt.

Bei der Mischung mit anderen Modi muss der GTIA während des Bi1dschirmaufbaus durch sog. Displaylist-Interrupts umgeschaltet werden. Im vorliegenden Fall werden die Bits 6 und 7 des Schattenregisters GPRIOR ausgeblendet, dieser Wert in das Hardwareregister PRIOR geschrieben und damit der GTIA-Modus abgeschaltet. Weiterhin wird der Zeichensatz-Zeiger wieder auf den normalen ROM-Zeichensatz gerichtet. Zum besseren Verständnis dieses Vorganges ist nachfolgend ein Assemblerlisting des DLI-Maschinenprogrammes abgedruckt, welches im obigen BASIC-Programm (Listing 3) als DATA–Zeilen enthalten ist.

```
0100 ;*****************************************
0110 ;Displaylist Interrupt
0120 ;fuer Textfenster in GTIA-Modi
0130 ;*****************************************
0140 ;
0150 GPRIOR = 623    Schattenregister von PRIOR
0160 PRIOR  = $D01B   Bit 6/7: GTIA-Modi
0170 WSYNC  = $D40A   Warten auf Hor.-Sync.
0175 CHBASE = $D409  Zeichensatz-Adressregister
0180 ;
0190        *=  $0600   in PAGE 6
0200 ;
0210 DLI    PHA         Akkuinhalt retten
0215        TXA         X-Register retten
0217        PHA
0220        LDA GPRIOR
0230        AND #$3F    Bit 6/7 ausblenden
0235        LDX #224    normaler Zeichensatz
0240        STA WSYNC   Zeilenende abwarten
0250        STA PRIOR   GTIA normal schalten
0255        STX CHBASE  regulaerer Z-Satz herstellen
0260        PLA         X-Reg. herstellen
0261        TAX
0262        PLA         Akku ebenfalls
0270        RTI         DLI beenden
0280 ;
```

Listing 4: DLI-Maschinenproqramm

Noch ein paar Worte zum Schluss. Die Verwendung der neuen Graphikmodi setzt den GTIA-Chip voraus. Bei älteren Geräten, die noch den CTIA enthalten, produzieren die obigen Programme keine vernünftige Graphik. Zum Trost sei gesagt: die meisten in Europa verkauften ATARI-Computer und alle neuen XL-Modelle enthalten den GTIA-Chip.

Textfenster in GRAPHICS 9/10/11

Das DLI-Maschinenprogramm des letzten Abschnittes kann man auch in den regulären GTIA-Graphikmodi 9-11 sehr gewinnbringend einsetzen, um auch dort das gewohnte vierzeilige Textfenster zu ermöglichen. Damit werden die GTIA—Modi gleich viel umgänglicher, denn nun können Sie auch einen PLOT oder DRAWTO-Befehl per Hand eingeben und das Ergebnis bewundern, ohne dass gleich wieder auf GR.O zurückgeschaltet wird.

Allerdings ist beim Aufruf der Graphikmodi ein Trick notwendig: Sie dürfen nicht GRAPHICS 9—11 direkt ansprechen, sondern rufen statt dessen GR.8 auf, installieren jetzt den DLI in der Display—List, und schalten erst dann mit Hilfe von GPRIOR (s. letzten Abschnitt) auf den entsprechenden GTIA-Modus um.

Im nachfolgenden Listing finden Sie ein Beispiel für die gerade geschilderte Vorgehensweise, das Unterprogramm ab Zeile 31000 können Sie direkt in eigenen Programmen als Ersatz für den GRAPHICS 9-11 Befehl verwenden.

```
10 REM *** GTIA-Modi mit TEXTFENSTER (Bsp.: GR.9) ***
20 REM Hier kann Ihr Programm stehen...
30 GR=9:GOSUB 31000:REM * GR gleich Graphikmodus (9-11)
100 SETCOLOR 4,1,0:REM * Farbe einstellen
110 FOR I=0 TO 15:REM * Bildschirmmuster zeichnen
120 COLOR I:PLOT I*5,0:DRAWTO I*5,159
130 NEXT I
160 LIST :STOP
180 REM
190 REM * ab hier Unterprogramm zum GR.9-11/TXT Aufruf...
200 REM Graphikmodus (9-11) muss in Variable GR enthalten sein
210 REM
31000 REM * GRAPHICS GR mit Textfenster einschalten *
31005 GOSUB 32000:REM * DLI-Maschinenprogramm
31010 GRAPHICS 8:REM * vorerst GRAPHICS 8 mit Textfenster
31020 DLIVEC=512:NMIEN=54286:SDLDTL=560
31030 DLIST=PEEK(560)+PEEK(561)*256:REM Displaylistadresse
31040 POKE DLIST+166,15+128:REM Bit fuer DLI setzen
31050 POKE DLIVEC,0:POKE DLIVEC+1,6:REM * DLI-Vektor richten
31060 POKE NMIEN,192:REM DLI wird aktiv
31070 GTIA=64:IF GR=10 THEN GTIA=128:REM * Wert fuer GPRIOR finden
31080 IF GR=11 THEN GTIA=192
31090 POKE 623,GTIA:REM * GTIA umschalten
31100 POKE 87,GR:REM * O.S. auf richtigen Graphik-Modus setzen
31110 RETURN
32000 REM * DLI-Maschinenprogramm einrichten
32010 RESTORE 32100
32020 FOR A=1536 TO 1558:READ D:POKE A,D:NEXT A
32090 RETURN
32100 DATA 72,138,72,173,111,2,41,63,162,224,141,10,212,141,27,208
32110 DATA 142,9,212,104,170,104,64
```

GTIA-Modi mit Texfenster

DISK I/O in Maschinensprache

"Was lange währt, wird einmal gut" — so könnte man BASIC-Programme beschreiben, die sich mit Daten-Ein/ Ausgabe auf Floppy beschäftigen. Wenn Geschwindigkeit gefragt ist, und das trifft schließlich in den meisten Fällen zu, dann muss man Assemblerpragramme oder wenigstens Maschinerunterprogramme in BASIC einsetzen. "Huch", sagen Sie, "ist das nicht schwierig?" Nein, ist es durchaus nicht! Ihr ATARI-Computer hat ein höchst komfortables Betriebssystem fest eingebaut, das Ihnen die an sich recht umfangreiche Aufgabe der I/O Programmierung sehr leicht macht. Sie werden sehen, die Sache ist fast so einfach wie in BASIC, und zusätzlich gibt es noch einige sehr nützliche Befehle, die Sie in BASIC nicht direkt ansprechen können.

Im Prinzip
gibt es zwei verschiedene Ebenen in denen Disk I/O-Operationen ablaufen können: Zum einen die Ein/Ausgabe auf Dateiebene, die sich mit der gewöhnlichen Filestruktur befaßt und zum anderen die 'Handlerebene', in der nur einfache Befehle, z.B. Sektor lesen oder schreiben, möglich sind. Wir wollen uns hier ausschließlich mit der Dateiebene befassen. Hier dreht sich alles um Files, mit deren Umgang Sie bestimmt schon vertraut sind. Files werden mit OPEN, CLOSE und Befehlen zum Schreiben und Lesen von Daten verwaltet. Diese ganzen Funktionen stehen Ihnen auch bei Assemblerprogrammierung zur Verfügung, man muss nur eben wissen wie man sie aufruft.

Grundlagen: Dateien-Files
was ist nun eigentlich ein 'File'? Speziell für die Disk ist damit einfach ein Datenblock gemeint, der im Disketteninhaltsverzeichnis unter seinem (File—)Namen eingetragen ist. Das folgende BASIC-Beispiel erzeugt ein kleines File:

```
10 OPEN#1,8,0,"D:BEISPIEL.BIN"
20 FOR 1= 0 T0 999:PUT#1,0:NEXT I
30 CLOSE#1
```

Wenn Sie jetzt im Inhaltsverzeichnis der Diskette nachsehen, so finden Sie das Beispiel unter dem Namen BEISPIEL.BIN verzeichnet. Zum Lesen eines solchen Files müssen Sie dann wie folgt vorgehen: Zuerst das File unter Bezug auf seinen Namen zum Lesen öffnen, dann Byte für Byte einlesen und das File wieder schließen, aber das ist Ihnen ja sicher bereits alles bekannt. Es gibt jetzt zwei Möglichkeiten: Entweder Sie wissen, wie lange das File tatsächlich ist, oder Sie machen es wie im folgenden Beispiel:

```
10 OPEN#1,4,0,"D:BEISPIEL.BIN"
20 TRAP 50
30 GET#1,D:?D:G0T0 30
50 CLOSE#1:TRAP 40000
```

Das File wird einfach solange gelesen, wie noch Bytes vorhanden sind. Wird über das Fileende hinaus gelesen, so bekommt man einen End-Of-File-Error (ERROR 136), der mit dem TRAP-Befehl aufgefangen wird. Spätestens wenn Sie die beiden Beispiele ausprobiert haben, merken Sie, wie wahr der Satz am Anfang dieses Abschnittes ist: Die BASIC Ein/Ausgabegeschwindigkeit ist extrem niedrig.

Wenn wir die eben geschilderten I/O-Operationen in Assembler schreiben wollen, so müssen wir uns zuerst ansehen. wie I/O-Befehle in Maschinensprache aufgerufen werden.

Die zentrale Ein/Ausqabe-Routine CIO

Die ganze Palette der I/0-Operationen auf Dateiebene wird von einer einzigen Betriebssystemroutine betreut, die über den Vektor CIOV ($E456) aufgerufen wird. Damit wir dieser Routine mitteilen können, welcher I/O—Befehl überhaupt auszuführen ist, müssen wir zuerst eine Art 'Formular' ausfüllen, in dem Befehlscode und eventuelle Parameter der I/0-Operation einzutragen sind. Acht derartiger Formulare', im Atari-Jargon IOCBs (Input/Output Control Block) genannt, sind im Speicher beginnend bei $340 vorhanden. Sehen wir uns zuerst den Aufbau eines IOCBs an:

IOCB Anf.	Feld Name	Funktion
+0	ICHID:	Handleridentifizierung, dieses Byte wird von CIO gesetzt. Wenn dieses Byte <>255, dann ist der IOCB frei.
+ 1	ICDNO:	Devicenumber. Nummer des Ein/Ausgabegerätes. Wichtig bei Diskettenbetrieb: Hier steht die Drivenummer. Auch dieses Byte wird von CIO während des OPEN-Befehls gesetzt.
+2	ICCMD:	Hier wird das Befehlsbyte eingetragen. Liste der Befehle siehe später.
+3	ICSTA:	Status-Byte, wird von CIO zurückgegeben. Wert >128 bedeutet Fehler.
+4	ICBAL:	Anfangsadresse des Filebuffers L/H. Hier wird
+5	ICBAH:	entweder die Adresse des Filenamens (bei OPEN...) oder des Bufferbereiches (bei Schreib/Lese-Befehl) eingetragen.
+6	ICPTL:	Spezieller Zeiger auf die Adresse (-1) der PUT—
+7	ICPTH:	CHARACTERS Routine des Handlers (L/H), wird normalerweise nicht benutzt.
+8	ICBLL:	Vor Aufruf von CIO muss die Länge des aktuellen
+9	ICBLH:	Buffers hier eingetragen werden, CIO gibt die Anzahl der tatsächlich gelesenen Bytes auch hier zurück
+ 10	ICAX1:	Zusätzliche Information, z.B. ob File zum Lesen
+ 11	ICAX2:	oder Schreiben geöffnet werden soll, ICAX1/2 darf nach 'OPEN' nicht mehr verändert werden.
+ 12	ICAX3:	Zusatz Informationen, die vom Handler benutzt werden, NOTE und POINT gebrauchen diese Bytes.
...		
...		
+ 15	ICAX6:	

Nachdem Sie das IOCB–Formular ausgefüllt haben (wie das im Detail gemacht wird erfahren Sie gleich anschließend), müssen Sie CIO aufrufen, natürlich nicht, ohne vorher im X–Register einen Hinweis hinterlassen zu haben, welcher der IOCBs zu benutzen ist. CIO ist auch noch anspruchsvoll, und möchte die IOCB–Nummer mundgerecht mit 16 multipliziert erhalten. Dahinter steckt ein einfacher Grund: Da jeder IOCB 16 Bytes lang ist, stellt das X–Register nun einen Zeiger auf den aktuellen IOCB dar, relativ gesehen zum IOCB Nummer Null.

Wenn Informationen zurückzugeben sind, so bekommen wir diese auch über den IOCB, natürlich mit Ausnahme der eigentlichen Daten, die sich in dem Bufferbereich befinden, der im IOCB angegeben wurde. Die Statusmeldung erhalten wir zur einfacheren Abfrage zusätzlich zum ICSTA–Feld auch im Y–Register zurück.

Da es insgesamt 8 dieser IOCB's gibt, könnten wir maximal 8 Files gleichzeitig offen halten. In der Praxis stimmt das nicht ganz, da einige IOCBs vom Betriebssystem selbst benötigt werden, z.B. IOCB#0 ist dauernd vom Screeneditor belegt, IOCB#6 und 7 werden im Zusammenhang mit Graphik–Modi und LOAD/SAVE und LPRINT Anweisungen gebraucht. Für Ihre Benutzung stehen damit noch die IOCBs 1–5 zur Verfügung.

a) File öffnen

Damit die Theorie nicht allzu trocken wird, können Sie sich gleich folgendes Beispiel ansehen: Ein Maschinenprogramm, das dieselbe Funktion wie der BABIC-Befehl

```
OPEN #2,8,0, "D:BEISPIEL.BIN"
```

ausführt, also das File BEISPIEL.BIN zum Schreiben öffnet.

Im Allgemeinen geht man so vor: Man definiert die IOCB-Felder nur für den IOCB Nummer Null (ab $340) und lädt die mit 16 multiplizierte Nummer des IOCBs (im OPEN die Nummer hinter '#') in das X–Register. Die Kommandos und Parameter der I/O-Operation werden dann X-indiziert relativ zu den IOCB#0 Definitionen in den gewünschten IOCB eingetragen, da das X-Register ja genau den Versatz des aktuellen IOCB-Feldes vom Feld des IOCBs#0 enthält. Zum Beispiels

```
ICCMD=$340+3   ;Feld für Befehlscode, IOCB#0
LDX  #2*16     ;IOCB #2 verwenden
STA   ICCMD,X  ;Befehl in IOCB#2 eintragen
```

Vorteile dieser Methode; Die IOCB-Labels brauchen nur einmal definiert werden, außerdem braucht nur das X–Register geändert werden, wenn man einen anderen IOCB benutzen will.

Schließlich wird CIO durch den Vektor CIOV ($E546) als Unterprogramm aufgerufen. Dazu muss das X-Register die IOCB–Nr. mal 16 enthalten, das haben wir aber schon eingangs erfüllt, so dass ein JSR CIOV genügt.

Im Y–Register erhalten wir eine Meldung über den Erfolg (oder Misserfolg) der I/O–Operation zurück, Ist Y kleiner 128, d.h. das N-Flag nicht gesetzt, so ist alles gut gelaufen. Bei Y–Werten größer oder gleich 128 ist ein Fehler aufgetreten, die Fehlercodes des Y–Registers sind identisch mit den BASIC–Fehlermeldungen, die im BASIC–Manual angegeben sind (s. auch Seite 65).

Hier ein Unterprogramm, welches die oben aufgeführte OPEN–Operation ausführt:

```
OPEN   LDX #2*16      IOCB-Nr. mal 16
       LDA #3          'OPEN'-Befehlscode
       STA ICCMB,X    in IOCB eintragen
       LDA #FNAME&255 Zeiger auf Filename. LSB
       STA ICBAL,X    als Bufferadresse eintragen
       LDA #FNAME/256 Zeiger, MSB
       STA ICBAH,X
       LDA #8          File zum Schreiben öffnen
       STA ICAX1.X     in AUX1 eintragen
;                      X enthält IOCB-Nr. mal 16
       JSR CIOV        CI0V=*E456
;                      Y enthält Statusmeldung!
       RTS
FNAME  .BYTE "D:BEISPIEL.BIN",$98   ($98 ist EOL)
```

Im einzelnen: Der Code für den Befehl 'OPEN' (03) wird in den IOCB eingetragen, in das Feld für die Bufferadresse wird ein Zeiger auf den Filenamen geschrieben, und schließlich die Richtung des (folgenden) Datentransfers durch AUX1 festgelegt. Zulässige Werte für AUX1 sind:

4 : File wird zum Lesen geöffnet.

8 : File zum Schreiben öffnen.

6 : Inhaltsverzeichnis zum Lesen öffnen

9 : Append: File wird zum Schreiben geöffnet, die Daten werden aber am Ende eines evtl. vorhandenen Files geschrieben.

12 : File zum gleichzeitigen Lesen und Schreiben öffnen, für Random-Access Files.

b) Daten schreiben

Daten können Sie in das zuvor geöffnete File mit folgendem Programm eintragen, welches in etwa dem BASIC-Befehl 'PUT#' entspricht. Der zu schreibende Wert muss im Akku stehen.

```
PUT  PHA           Wert merken
     LDX #2*16     IOCB-Nr. mal 16
     LDA #11       Befehl 'Put Characters'
     STA ICCMD,X   in IOCB eintragen
     LDA #0         nur einen Wert schreiben
     STA ICBLL,X    daher Bufferlange:=0
     STA ICBLH,X
     PLA           Wert holen
     JSR CIOV      und ablegen...
     CPY#0         Fehler? (s. Text!)
     BMI ERROR      ja, zur Fehlerroutine
     ...
```

Der "CPY #0'-Befehl kann entfallen, da die Flags von CIO bereits auf das Y–Register bezogen sind, so dass BMI alleine auch genügt, er wurde nur zur Verdeutlichung des Sachverhalts im Programm aufgenommen. Wir haben den 'Put Characters' Befehl beim letzten Beispiel in einer Spezialform angewandt, die nur die Ausgabe eines einzigen Bytes zulässt. Die Übergabe des Datums erfolgte dabei über den Akku.

c) Datenblöcke schreiben

Es ist aber auch möglich, statt einzelner Bytes ganze Datenblöcke mit einem I/O—Befehl abzuspeichern. Nehmen wir einmal an, Sie wollen den Bereich $8000 bis $9FFF (8 Kbyte Länge, $2000 Bytes) abspeichern, so könnten Sie das mit folgendem Programmteil erledigen (OPEN wie unter Punkt a vorausgesetzt):

```
  PUTBLK LDX #2*16      IOCB-Nr. mal 16
         LDA #11 Befehl 'Put Characters'
         STA ICCMD,X    in IOCB
         LDA #0         Anfangsadresse LSB
         STA ICBAL,X    Bufferadresse LSB
         LDA #$80       Anfangsadresse MSB
         STA ICBAH,X    Bufferadresse MSB
         LDA #0         Länge Datenblock LSB
         STA ICBLL,X    Bufferlänge LSB
         LDA #$20       MSB Länge ($2000)
         STA ICBLH,X    Bufferlänge MSB
         JSR CIOV       Datenblock speichern
         BMI ERROR      Fehler --->
         ...
```

d) Daten lesen

Mit diesem Programmteil können Sie den in c) aufgezeichneten Datenblock wieder einlesen. Vorher muss selbstverständlich ein passender OPEN-Befehl gegeben werden, in dem AUX1 mit dem Wert 4 (öffnen zum Lesen) besetzt wird (ICAX1:=4).

```
GETBLK LDX #2*16 IOCB-Nr. mal 16

         LDA #7         Befehl 'Get Characters'
         STA ICCMD,X    in IOCB
         LDA #0         Anfangsadresse LSB
         STA ICBAL,X    Bufferadresse LSB
         LDA #$80       Anfangsadresse MSB
         STA ICBAH,X    Bufferadresse MSB
         LDA #0         Länge Datenblock LSB
         STA ICBLL,X    Bufferlänge LSB
         LDA #$20       MSB Länge ($2000)
         STA ICBLH,X    Bufferlänge MSB
         JSR CIOV       Datenblock lesen
         BMI ERROR      Fehler --->
         ...
```

e) Einzelne Bytes lesen

Das Beispiel unter Punkt b) zum Lesen eines einzelnen Bytes muß etwas modifiziert werden:

```
         GET LDX#2*16   IOCB-Nr. mal 2
         LDA #7         'Get Character' Befehl
         STA ICCMD,X    in IOCB
         LDA #0         Bufferlänge:=0
         STA ICBLL,X
         STA ICBLHxX
         JSR CIOV       Ein Byte lesen
;                       Byte ist im Akku
         BMI            ERROR Fehler? --->
         ...
```

Dieses Codestück entspricht dem GET#-Befehl in BASIC.

f) File schließen

Das ist die einfachste Übung: Sie brauchen nur den Code für den CLOSE-Befehl in den IOCB zu schreiben und CIO aufzurufen. Die restlichen Parameter brauchen Sie nicht zu besetzen, da sie von CIO nicht beachtet werden.

```
  CLOSE  LDX #2*16      IOCB-Nr. mal 16
         LDA #12        Code für  CLOSE
         STA ICCMD,X    in IOCB
         JSR CIOV       Close ausführen
         ...
```

Zum Schluß

dieses Abschnittes ist noch zu sagen, dass Sie die beschriebenen Methoden zum Datentransfer mit CIO nicht nur im Zusammenhang mit Disk-Files verwenden können. Schreiben Sie anstatt der Filespezifikation in OPEN-Befehl nur den Kürzel eines I/O-'Gerätes' z.B. 'C:' für die Cassette oder 'P:' für den Drucker, so können Sie auch Daten auf andere Geräte übertragen. Wichtig ist hier noch anzumerken, dass es zusätzlich noch Befehle zum Übertragen von satzorientierten Dateien gibt. Auf diese Art des Datentransfers soll hier nicht näher eingegangen werden, soviel sei aber gesagt, dass es sich dabei um die Übertragung von RECORDS handelt, die jeweils mit einem EOL-Zeichen ($9B) abgeschlossen sind. Verwendung findet diese Methode z.B. bei den BASIC INPUT und PRINT Befehlen. Die Codenummern der zugehörigen CIO-Anweisungen 'GET/PUT RECORD' können Sie der nachfolgenden Tabelle entnehmen.

Tabelle der wichtigsten IO-Befehle

Befehl	ICCMD	ICBAL/ ICBAH	ICBLL/ ICBLH	ICAX1	Befehls Label
OPEN	3	Zeiger auf Filenamen	. /.	Modus	COPN
CLOSE	12	. /.	. /.		CCLOSE
STATUS	13	Zeiger auf Filenamen	. /.		CSTAT
GET CHARACTERS	7	Buffer-adresse	Buffer-länge		CGBINR
PUT CHARACTERS	11	Buffer-adresse	Buffer—länge		CPBINR
GET RECORD	5	Buffer-adresse	Max. B. -länge		CGTXTR
PUT RECORD	9	Buffer-adresse	Max. B--länge		CPTXTR

Wichtige Fehlermeldungen

128: Break-Taste gedrückt

129: gewünschter IOCB bereits belegt

130: Device ('Gerät') existiert nicht, z.B. D: vergessen

133: File nicht geöffnet (OPEN falsch oder fehlt)

134: keine gültige IOCB-Nummer

136: End-of-File, über Fileende hinausgelesen

137: Record paßt nicht in Buffer: grösseren Buffer wählen

138: Timeout, Operation hat zu lange gedauert

139: Device NAK, unausführbarer Befehl, z.B. Sektor 0

144: Device done, z.B. zerstörter Sektor

162: Diskette ist voll

164: FMS—Sektorverkettung stimmt nicht.

167: File verriegelt, mit DOS 'G' entriegeln.

170: File nicht gefunden

BLS - BASIC und Binärfiles

Die im letzten Abschnitt beschriebenen GET/PUT CHARACTERS Befehle zum schnellen Laden und Speichern von Datenblöcken können in BASIC leider nicht direkt benutzt werden. Die Assemblerroutine BLS (Binär Laden und Speichern) hilft Ihnen jedoch geschickt über diesen Mangel hinweg. Mit Hilfe von BLS können Sie ganze Speicherbereiche unter Angabe von Anfangsadresse und Länge auf Disk ablegen oder von auch Disk laden. Eine ideale Sache, wenn Hi—Res Bilder, Zeichensätze, Daten für Player-Missile Graphik oder auch Maschinenprogramme innerhalb eines BASIC—Programmes von Disk geladen oder auch darauf aufgezeichnet werden sollen.

Trick 17

Um das Maschinenunterprogramm so kurz wie möglich zu halten, wird ein Trick angewendet: OPEN- und CLOSE-Befehle werden in BASIC ausgeführt, das Maschinenprogramm übernimmt ausschließlich die Aufgabe des tatsächlichen Datentransfers. Damit spart man sich die Übergabe des Filenamens an das Maschinenprogramm und gewinnt außerdem die Flexibilität, zwischen OPEN, Datenblocktransfer und CLOSE noch einige Bytes sozusagen 'per Hand' aus dem File lesen oder in das File schreiben zu können. So wäre es ohne weiteres möglich, dass man z.B. nach dem Abspeichern eines Hi—Res Bildes die Inhalte der Farbregister mit Hilfe des PUT#-Befehles ablegt, beim Laden des Bildes diese wieder mit GET# abfängt und in die zugehörigen Farbregister schreibt.

Das BLS—Programm ist in den nächsten Beispielen in DATA—Zeilen enthalten, zum besseren Verständnis finden Sie ein Assemblerlisting am Ende dieses Abschnittes.

BLS wird wie folgt aufgerufen:

X=USR(BLS,<Flag>,< Anfangsadresse>.<Länge des Blockes»

wobei BLS=1536 die Anfangsadresse des Maschinenprogrammes ist (Page 6), Flag muss zum Lesen gleich 0 sein, zum Schreiben des Datenblockes eine Zahl ungleich 0, die Parameter Anfangsadresse und Länge sind aus dem letzten Abschnitt bereits hinreichend bekannt. Vor dem Aufruf von BLS muss wie oben schon erwähnt ein passender OPEN-Befehl erfolgt sein:

OPEN#3,4,0,"D:..." beim Lesen

OPEN#3,8,0,"D:..." zum Schreiben

eines Files. Wichtig ist dabei auch, dass immer der IOCB#3 verwendet wird (also immer OPEN#3...), da das Maschinenprogramm für diesen I/O-Kanal ausgelegt ist.

Maschinenprogramme in BASIC laden

Günstig erweist sich die Aufteilung von BASIC zu Assemblerprogramm auch dann, wenn man Maschinenprogramme, die z.B. mit der Editor/Assembler Cartridge oder dem DOS-Befehl 'K' (Binary Save) abgespeichert wurden, innerhalb eines BASIC-Programmes einlesen will. Diese Art von Files (sog. Binary Load Files) enthalten nämlich vor dem Datenblock noch einige Steuerinformationen: zwei Kennbytes (=$FF), die Anfangs- und Endadresse im normalen 6502 L/H—Format.

Die beiden Kennbytes, Anfangs- und Endadresse müssen in BASIC mit GET#- Befehlen abgefangen und in die passenden Informationen für die BLS-Routine umgerechnet werden. Das folgende BASIC-Programm zeigt, wie's gemacht wird:

```
100 REM **** Maschinenprogramm in BASIC laden ****
110 GOSUB 32000:REM * Maschinenprogramm initialisieren
120 OPEN #3,4,0,"D:FILENAME.EXT":REM *  Hier Filenamen einsetzen
130 GET #3,X1:GET #3,X2:REM *       Kennbytes lesen
140 IF X1<>255 OR X2<>255 THEN CLOSE #3:? "Kein Binaer-File!":STOP
150 GET #3,AL:GET #3,AH:ANFADR=AL+AH*256:REM * Anfangsadresse
160 GET #3,AL:GET #3,AH:ENADR=AL+AH*256:REM *  Endadresse
170 LAENGE=ENADR-ANFADR+1:REM *     Anzahl der Bytes
180 X=USR(BLS,0,ANFADR,LAENGE):REM *File einlesen
190 CLOSE #3
200 IF X>128 THEN ? "DISK-FEHLER: ";X
290 END
32000 REM * BLS einrichten
32010 S=0:RESTORE 32100
32020 FOR A=1536 TO 1576:READ D:POKE A,D:S=S+D:NEXT A
32030 IF S<>4119 THEN ? "DATEN-FEHLER!":STOP
32040 BLS=1536
32090 RETURN
32100 DATA 104,162,48,104,160,7,104,240,2,160,11,152,157,66,3,104
32110 DATA 157,69,3,104,157,68,3,104,157,73,3,104,157,72,3,32,86,228
32120 DATA 132,212,169,0,133,213,96
```

Beispiel 1: Laden eines Maschinenproqrammes in BASIC

Ebenfalls ist es natürlich kein Problem, mit der BLS-Routine Hi-Res Bilder (z.B. GR.8) zu laden oder auch abzuspeichern. Sie brauchen dazu nur die Anfangsadresse des Bildschirmspeichers (steht in 88/89 dez. L/H) und die Länge des GR.8 Bildes (40 Bytes pro Zeile mal 192 Zeilen = 7680 Bytes). Hier zuerst ein Programm mit dem Sie einen Demo-Screen abspeichern können:

```
100 REM **** Demo: GR.8- Bilder abspeichern  ****
110 GOSUB 32000:REM **** Maschinenprogramm
115 GRAPHICS 8+16:REM ** ganzer Gr.8 Bildschirm
120 SETCOLOR 2,0,0:REM * Hintergrund schwarz
125 COLOR 1:PLOT 159,95:REM GR.8 Bild erzeugen
130 FOR A=0 TO 50:DRAWTO 319*RND(1),191*RND(1):NEXT A
135 SCREEN=PEEK(88)+PEEK(89)*256:REM * Bildschirmadresse
140 LAENGE=7680:REM **** Laenge Gr.8/7+ Bild
150 OPEN #3,8,0,"D:TEST.PIC":REM * Hier Filenamen einsetzen
160 X=USR(BLS,1,SCREEN,LAENGE)
170 CLOSE #3
180 IF X>128 THEN ? "DISK-FEHLER: ";X
190 END
32000 REM * BLS einrichten
32010 S=0:RESTORE 32100
32020 FOR A=1536 TO 1576:READ D:POKE A,D:S=S+D:NEXT A
32030 IF S<>4119 THEN ? "DATEN-FEHLER!":STOP
32040 BLS=1536
32090 RETURN
32100 DATA 104,162,48,104,160,7,104,240,2,160,11,152,157,66,3,104
32110 DATA 157,69,3,104,157,68,3,104,157,73,3,104,157,72,3,32,86,228
32120 DATA 132,212,169,0,133,213,96
```

Beispiel 2: Abspeichern eines GR.8 Bildes mit BLS

Mit dem nächsten Programm können Sie das eben abgespeicherte Hi-Res Bild wieder einlesen, selbstverständlich können Sie damit auch Bilder, die mit einem Graphik-Programm erzeugt wurden, in eigenen Programmen zeigen.

```
100 REM **** GR.8- Bilder laden ****
110 GOSUB 32000:REM **** Maschinenprogramm
120 GRAPHICS 8+16:REM ** ganzer Gr.8 Bildschirm
130 SETCOLOR 2,0,0:REM * Hintergrund schwarz
135 SCREEN=PEEK(88)+PEEK(89)*256:REM * Bildschirmadresse
140 LAENGE=7680:REM **** Laenge Gr.8/7+ Bild
150 OPEN #3,4,0,"D:TEST.PIC":REM * Hier Filenamen einsetzen
160 X=USR(BLS,0,SCREEN,LAENGE)
170 CLOSE #3
180 IF X>128 THEN ? "DISK-FEHLER: ";X
190 GOTO 190:REM ******* um Graphic nicht zu zerstoeren
32000 REM * BLS einrichten
32010 S=0:RESTORE 32100
32020 FOR A=1536 TO 1576:READ D:POKE A,D:S=S+D:NEXT A
32030 IF S<>4119 THEN ? "DATEN-FEHLER!":STOP
32040 BLS=1536
32090 RETURN
32100 DATA 104,162,48,104,160,7,104,240,2,160,11,152,157,66,3,104
32110 DATA 157,69,3,104,157,68,3,104,157,73,3,104,157,72,3,32,86,228
32120 DATA 132,212,169,0,133,213,96
```

Beispiel 5: GR.8 Bild mit BLS laden

Auf der nächsten Seite finden Sie das kommentierte Assemblerlisting des BLS-Programmes. Eine kurze Beschreibung dieses recht universellen Programmes folgt:

Programmübersicht BLS

180	Nummer des benutzten IOCBs * 16
190-200	Codes für 'GET/PUT CHARACTERS'-Befehl.
220-260	Definitionen für IOCB-Parameter: Befehl, Bufferadresse, Bufferlänge.
280	CIO-Einsprungvektor
310	Speicherbereich f. Rückgabe des USR-Ergebnisses
390	Programm ist in PAGE 6 gelegt.
400	Anzahl der USR-Argumente vom Stack nehmen
410	X-Register mit IOCB-Offset laden
420-500	Flag= 0 -> 'GET CHARACTERS'-Befehl in IOCB eintragen Flag<>0 -> 'PUT CHARACTERS'
510-540	Bufferadresse vom Stack in IOCB übertragen
550-580	Bufferlänge vom Stack in IQCB übertragen
600	CIO-Aufruf, Status kommt im Y-Register zurück
620-640	Status in USR-Ergebnis zurückgeben

```
0100 ;*******************************************
0110 ; BLS - Binaer-File von BASIC laden/speichern
0120 ;
0130 ;*******************************************
0140 ;
0150 ;
0160 ; CIO-Definitionen
0170 ;
0180 IOCB3  = $30     IOCB Nummer 3 benutzen
0190 CGBINR = $07    CIO-Befehl: Characters lesen
0200 CPBINR = $0B    CIO-Befehl: Characters schreiben
0210 ;
0220 ICCMD  = $0342   Hier wird CIO-Befehl abgelegt
0230 ICBAL  = $0344   Bufferadresse Low-Byte
0240 ICBAH  = $0345   Hi-Byte
0250 ICBLL  = $0348   Bufferlaenge Low-Byte
0260 ICBLH  = $0349   Hi-Byte
0270 ;
0280 CIOV   =  $E456   CIO-Einsprung
0290 ;
0300 ;
0310 FR0    =   $D4     Ergebnis von USR-Befehl
0320 ;
0330 ;*************************************************
0340 ;USR-Routine: LOAD/SAVE eines Binaer Files
0350 ;Aufruf: X=USR(1536,Flag,Startadresse,Laenge)
0360 ;Flag: 0:= LOAD, 1:= SAVE
0370 ;*************************************************
0380 ;
0390        *=  $0600   in PAGE 6 (wo auch sonst?!)
0400        PLA         Anzahl der Argumente
0410        LDX #IOCB3  IOCB Nr. 3 wird benutzt
0420        PLA         Hi-Byte Flag Lesen/Schreiben
0430        LDY #CGBINR Vorbesetzung: Chr. lesen
0440        PLA         Flag:Lesen=0, Schreiben=1
0450        BEQ BCMD    Flag=0, Lesen bleibt -->
0460 ;
0470        LDY #CPBINR sonst Chr. schreiben
0480 ;
0490 BCMD   TYA        Befehl in Akku
0500        STA ICCMD,X Befehl in IOCB
0510        PLA         Anfangsadresse Hi-Byte
0520        STA ICBAH,X in Bufferadresse eintragen
0530        PLA         Lo-Byte
0540        STA ICBAL,X
0550        PLA         Laenge des Buffers Hi-Byte
0560        STA ICBLH,X
0570        PLA         Lo-Byte
0580        STA ICBLL,X
0590 ;
0600        JSR CIOV    Zentrale I/O-Routine aufrufen >>>
0610 ;
0620        STY FR0     Status in USR-Ergebnis
0630        LDA #0      an BASIC zurueckgeben
0640        STA FR0+1
0650        RTS
0660 ;
```

Neue Befehle für OS/A+ DOS

Das OS/A+ DOS von Optimized Systems Software kann seine Verwandtschaft mit CP/M, einem der populärsten Betriebssysteme für (Z-80) Mikroprozessoren nicht verleugnen. Und gerade dies macht es zu einem universellen und leistungsfähigen Werkzeug für den Programmierer. Die Eingabe ist kommandoorientiert, was zwar dem Anfänger schwieriger als die Auswahl aus einem Menü (Atari-DOS) erscheint, für den fortgeschrittenen Benutzer aber eher Vorteile bringt, man denke dabei nur an die Möglichkeit der Batch-Verarbeitung (an anderer Stelle in diesem Buch beschrieben), Weiterhin ist die Struktur des OS/A+ DOS für Erweiterungen offen, die Befehls1iste kann sehr leicht vom Benutzer vergrößert werden. Dadurch können Sie sich OS/A+ auf Ihre ganz individuellen Bedürfnisse zuschneidern.

Befehlsarten des OS/A+ DOS

Zwei Arten von Befehlen lassen sich beim OS/A+ DOS unterscheiden: da sind zum Einen die 'intrinsic'-Befehle, die fester Bestandteil des DOS und daher zu jeder Zeit aufrufbar sind (z.B. DIR, LOAD, SAVE). Zum Anderen gibt es die sogenannten 'extrinsic'—Befehle, die nicht im Speicher resistent sind, sondern bei Bedarf von der Diskette nachgeladen werden. Zu dieser zweiten Befehlsart zählen z.B. die COPY, DUPDSK und INIT-Kommandos.

Beispiel INIT

Was geht nun beim Aufruf eines derartigen 'extrinsic' Befehles, nehmen wir stellvertretend INIT, vor? Nachdem Sie 'INIT' eingetippt haben, stellt OS/A+ zuerst einmal anhand seiner Befehlsliste fest, daß es sich dabei nicht um einen seiner 'intrinsic'—Befehle handelt. Die Suche wird dann im Disketteninhaltsverzeichnis fortgesetzt, und schließlich wird, vorausgesetzt, daß die richtige Diskette eingelegt war, das File INIT.COM gefunden. Dieses File wird nun geladen und, solange es keine eigene Aufrufadresse mitbringt, an seinem Programmanfang gestartet. Jetzt sind wir an dem Punkt angelangt, an dem Ihnen das INIT-Menü am Bildschirm präsentiert wird.

Sie sehen: Die Erweiterung des Befehlsvorrates von OS/A+ ist recht einfach: Sie brauchen nur ein Maschinenprogramm, das übrigens natürlich auch von einem BASIC—Compiler stammen kann, unter dem gewünschten Befehlsnamen mit dem Extender .COM abzuspeichern, und schon ist es Bestandteil von OS/A+.

Übernahme von Parametern

Nun wäre es doch recht einfältig, nur Zusatzbefehle zu schreiben, die Ihre Eingaben wie der INIT-Befehl per Menü *oder* durch eine Art von INPUT-Befehl anfordern müßten. Nein - das OS/A+ DOS bietet Ihnen eine sehr viel elegantere Möglichkeit: Sie können Parameter und Zusatzinformationen direkt aus der Eingabezeile abholen, in der der Befehl aufgerufen wurde.

Der OS/a+ Copy-Befehl gibt ein Beispiel, wie es funktioniert. Nehmen wir einmal an, Sie wollen ein File kopieren und geben dazu folgendes ein:

```
D1:COPY D1:FILE2.TST D1:FILE2.TST <Return>
```

Anfänglich passiert jetzt das gleiche wie oben beim INIT—Befehl. COPY gehört ebenfalls nicht zu den 'intrinsic' Kommandos, daher wird das File COPY.COM von der Diskette geladen. Jetzt passiert's: COPY holt sich seine Parameter, die beiden Filenamen, direkt aus der Aufrufzeile.

Sie können diese Art der Parameterübergabe auch genauso in Ihren eigenen Programmen verwenden. Stellen Sie sich vor, Sie hätten ein Hardcopy-Programm (z.B. HARDCDPY.COM) geschrieben, das ein Bildfile auf dem Drucker ausgibt. Mit der in diesem Abschnitt beschriebenen Methode können Sie dem Programm ganz elegant mitteilen welches File auszudrucken ist:

```
D1:HARDCOPY D1:BILD1.PIC
```

Erkennen Sie jetzt, welche interessanten Möglichkeiten darin stecken?

Der Kommandobuffer

OS/A+ speichert die komplette Eingabezeile (inklusive des Befehlswortes) in seinem Befehlsbuffer und setzt einen Bufferzeiger dorthin, wo für OS/A+ die Bearbeitung des Befehls zu Ende ist, konkret nach dem Ende des Befehlswortes.

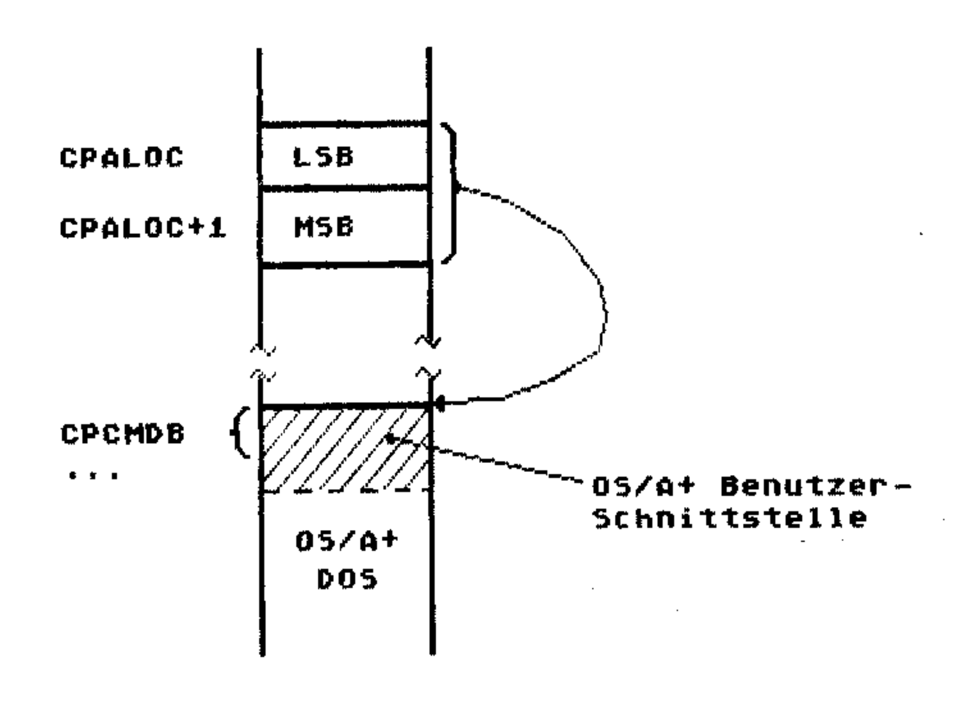
Der OS/A+ Befehlsbuffer

Der Ausdruck für die Anfangsadresse des Buffers sieht recht kompliziert aus, und dafür gibt es einen triftigen Grund: Kompatibilität. Ihre Programme sollen schließlich nicht nur mit der derzeitigen Version des OS/A+ DOS laufen, sondern auch mit älteren oder zukünftigen Versionen. Die Programmierer von OS/A+ haben dies bedacht, und eine universelle Schnittstelle zum Benutzer vorgesehen: alle Variablen, Programmaufrufe und Bufferadressen verstehen sich relativ zur Anfangsadresse von OS/A+, die Sie in den Speicherzellen CPALOC und CPAL0C+1 ($0A/0B, L/H) finden, den Benutzern von ATARI-DOS vielleicht eher unter der Bezeichnung DOSVEC geläufig.

Anmerkung

Die im folgenden verwendeten Variablennamen und Konstantenbezeichnungen entsprechen den im SYSEQU.ASM File benutzten Labels.

SYSEQU.ASM finden Sie auf der OS/A+ Systemdiskette. Sie können sich bei eigenen Assemblerprogrammen viel Tipparbeit ersparen, wenn Sie dieses File per INCLUDE in Ihr Programm einbinden. Sie haben dann gleich alle wichtigen Variablen und Konstanten (auch die IOCB-Konstanten) in Ihrem Programm enthalten.



Benutzerschnittstelle zum OS/A+ DOS

Die Adresse des Befehlsbuffers können Sie ermitteln, indem Sie zur Basisadresse im Vektor CPALOC den Offset des Buffers (CPCMDB, $40) addieren, Das Ergebnis sollte in der Zeropage abgelegt werden, um es als indirekte Basisadresse zum Lesen des Buffers verwenden zu können:

```
REG1L = $CE     Hilfsregister Zeropage CLSB)
REG1H = $CF     MSB
...
CLC             Addition   vorbereiten
LDA CPALOC      LSB  DOS-Anfangsadresse
ADC #CPCMDB     Offset Befehlsbuffer (*40)
STA REG1L       in Hilfsregister
LDA CPAL0C+1    MSB DOS-Anfang
ADC #0          evtl. Übertrag
STA REG1H       in MSB Hilfsregister
...
```

Jetzt benötigen Sie noch den Wert des Bufferzeigers, den Sie mit einem Offset von CPBUFP ($0A) erreichen können.

```
LDY #CPBUFP     Offset Bufferzeiger
LDA (CPALOC),Y  Wert des Bufferzeigers
TAY             als neuen Index verwenden
```

Mit Hilfe der Basisadresse in REG1L/H und des Index im Y-Register können Sie den Befehlsbuffer Zeichen für Zeichen lesen:

```
LDA <CPALOC),Y  Zeichen aus Befehlsbuffer
INY             nächstes Zeichen
...
```

Mehr Komfort

Bei den Parametern dürfte es sich wohl am häufigsten um Filenamen und Filespezifikationen handeln. Das OS/A+ DOS berücksichtigt diesen Umstand und stellt dem Programmierer freundlicherweise gleich ein Unterprogramm 'GET FILENAME' zur Verfügung, mit dem ein Filename aus dem Befehlsbuffer, beginnend ab dem momentanen Wert des Bufferzeiqers, in einen gesonderten Filename—Buffer übernommen werden kann. Dabei wird komfortablerweise eine eventuell fehlende Gerätebezeichnung mit dem aktuellen Default-Drive (normal 'D1:') ergänzt, und der Filename mit einem EOL ($9B) abgeschlossen. Die so bearbeitete Filespezifikation können Sie direkt in einem OPEN—Befehl verwenden. Gleichzeitig wird der Bufferzeiger weitergeführt, so dass ein weiterer Aufruf von 'GET FILENAME"' den nächsten Namen (falls vorhanden!) bringen würde.

Die Einsprungadresse (CPGNFN, $03) dieses Unterprogrammes ist, genau wie die anderen Schnittstellenadressen, relativ zum DOS-Anfang zu verstehen. Leider verfügt der 6502 über keinen indirekten Unterprogrammaufruf, so dass man sich hier anders behelfen muss. Der Einsprungvektor wird, wie auch im obigen Beispiel, wieder durch Addition des CPALOC—Vektors mit dem 'GET FILENAME'-Offset ermittelt, und dann als Adresse eines JMP-Befehles eingetragen (selbstverändernder Code!).

```
GETADR CLC            diese UP einmal Aufrufen
       LDA CPALOC     LSB DOS-Anfang
       ADC #CPGNFN    Offset Einsprungadresse
       STA GETFN+1    in JHP eintragen
       LDA CPAL0C+1   MSB DOS-Anfang
       ADC #0         evtl. Übertrag
       STA GETFN+2    MSB in JMP-Befehl
       RTS            Fertig!
;
GETFN  JHP 0          Adresse wird eingetragen
```

Nachdem einmal das obige Unterprogramm GETADR aufgerufen wurde, bringt jeder JSR GETFN eine Filespezifikation aus dem Befehlsbuffer in den Filenamebuffer. Der Programmablauf führt dabei über den erzeugten JMP-Befehl zur eigentlichen Einsprungadresse, das RTS steht dann im GET FILENAME Unterprogramm von OS/A+.

Jetzt fehlt nur noch eine Möglichkeit, wie man überprüfen kann, ob überhaupt ein Filename vorhanden war. Das lässt sich recht einfach mit Hilfe des Bufferzeigers machen, man braucht nur seinen Wert zu merken und nach dem Aufruf von GETFN zu prüfen, ob er sich verändert hat. Folgendes Unterprogramm zeigt, wie es programmiert werden könnte:

```
GETNAM LDY #CPBUFP    Offset Bufferzeiger
       LDA (CPALOC),Y Wert des Pufferzeigers
       PHA            einstweilen merken
       JSR GETFN      Filename holen
       PLA            alten Bufferzeiger holen
       CMP (CPALOC),Y = neuen Wert?
       BEQ KEINFN     ja, es war kein FN da —>
       ...
```

War der SETFN Aufruf erfolgreich, d.h. hat sich der Bufferzeiger verändert, dann können Sie den kompletten Filenamen aus dem Filenamebuffer (Offset vom DOS-Anfang: CPFNAM, $21) abholen. Hier ein Beispiel für ein entsprechendes Programm:

```
       LDY #CPFNAM    Offset Filenamebuffer
       LDX #0         Bytezähler auf 0 setzen
       NXTCHR LDA     (CPALOC,Y Zeichen aus FN-Buffer abholen
       STA FNAME,X    in eigenen Buffer ablegen
       CMP #EOL       Schon fertig? (E0L=*9B)
       BEQ FERTIG     ja --->
       INY            Zeiger auf nächstes Zeichen
       INX            Bytezähler
       CPX #16        schon 17-tes Zeichen?
       BNE NXTCHR     nein, nächstes Zeichen --->
       JMP ERROR      Kein EOL? Sollte nicht vorkommen!
FERTIG ...
;
FNAME *=*+16          16 Bytes Buffer für Filenamen
```

Ein Beispiel zur Verwendung dieser Technik können Sie dem APPEND-Programm im nächsten Abschnitt entnehmen.

Wohin mit den Zusatzbefehlen im Speicher?

Die einfachste Möglichkeit bietet hier, wie so oft, die PAGE 6. Leider kann man dort nur Programme bis zu einer Länge von 256 Bytes ablegen, für viele Anwendungen zu wenig. In diesen Fällen müssen Sie auf den Speicherbereich oberhalb von OS/A+ ausweichen, was einige Nachteile bringt:

1.) Die Obergrenze des von OS/A+ DOS benutzten RAMs ist bei verschiedenen Versionen höchst unterschiedlich. Wenn Ihr Programm mit allen OS/A+ DOS Versionen lauffähig sein soll, dann sollten Sie es nicht zu weit vorne beginnen lassen. Als Richtwert kann hier $2C00 angeben werden.

2.) In diesem Speicherbereich können sich Programme befinden, die dann durch den DOS-Befehlsaufruf zerstört werden. Vergewissern Sie sich daher, ob sich vor einem 'extrinsic'-Befehlsaufruf nichts mehr Wesentliches im Speicher befindet.

Runadresse

Das .COM—File wird von OS/A+ mit einem Unterprogrammsprung auf seine physikalische Anfangsadresse gestartet. Möchten Sie eine andere Einsprungadresse festlegen, so können Sie das erreichen, indem Sie Ihrem Programm noch eine RUN-Adresse mitgeben. Schreiben Sie, am besten am Schluß Ihres Programmes, folgendes:

```
*= $02E0        Hier wird RUN-Adresse abgelegt
.WORD START     START ist Einsprungadresse Ihres Programmes
```

Beenden Sie Ihr Programm mit einem RTS, so geht die Kontrolle wieder auf OS/A+ zurück.